Mach: A Foundation for Open Systems

A Position Paper

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

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

1. Introduction

Operating systems have become one of the most hotly contested battlegrounds for "open system standards". Various national, international and industry groups are attempting to define, implement and ultimately convince users to buy new "open" computing environments. Most of these efforts have centered around versions of the UNIX[1] [9] operating system, but there is no consensus among industrial groups as to which "version" of UNIX is ultimately the correct basis for a open system standard. Already two major industrial organizations, the Open Software Foundation and UNIX International, have endorsed two rather different UNIX implementations: IBM's AIX and AT&T's System V.4. Non-UNIX systems such as the Macintosh OS, MS-DOS and OS/2 are also seen as "standards" by various hardware and software developers.

This divergence of systems and standards raises fundamental issues about the strategy which should be employed in the development of open operating systems. It has become increasingly important to provide support within a single computing environment for "multiple standards", i.e. multiple operating system environments which may be tailored to different vendor or user needs. Moreover, manufacturers frequently need to provide customers with continuing access to proprietary operating systems developed during the 1960's and 1970's (e.g. VMS, MVS and MS-DOS).

Traditional systems such as UNIX or VMS are implemented "all in one piece" with knowledge about the basic system structure spread throughout. They are poorly suited to the compatible support of multiple operating system environments. As a result, large computer manufacturers are commonly forced to support several completely distinct operating system groups: one for OS/2, one for UNIX and one or more for proprietary systems -- each at considerable expense. Ballooning software development costs translate into increased cost for the user and delays in the introduction of new features and new applications.

[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. An Alternative Approach to OS Organization

An alternative approach to building an entire operating system is to separate those parts of the operating system which control the basic hardware resources -- often called the operating system "kernel" -- from those parts of the operating system which determine the unique characteristics of an operating system environment, for example a particular file system interface. The advantage of this scheme is that it can allow more than one operating system environment to be implemented on the same hardware/software base so that machine dependent software need be written only once for each new architecture.

The Carnegie Mellon University Mach kernel [1] is an example of this layered approach to operating system design. Mach is a multiprocessor kernel that incorporates in one system a number of key facilities that allow the efficient implementation of those functions necessary to support binary compatibility with existing operating system environments. These mechanisms are intended not simply as extensions to normal operating system facilities but as a foundation upon which UNIX and other operating system facilities can be built.

We believe that such a design is very well suited to the implementation of modern systems, and that it can yield significant improvements in terms of structure, without sacrificing performance. Furthermore, it also allows, at least potentially, for more than one operating system environment to be supported in "native mode" simultaneously on the same hardware. To demonstrate the practical applicability of this concept, and to evaluate its strengths and weaknesses, we are currently working on three separate approaches to implementing a 4.3 BSD [3] environment within the Mach framework.

3. Mach Features Supporting OS Emulation

Mach provides an unusually flexible execution environment for both system and user applications. It exposes the management of CPU, communication, virtual memory and secondary storage resources in a way that allows system applications such as database management facilities to use those resources efficiently.

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) [10],
- architecture independent virtual memory management (VM) [7],
- integrated IPC/VM support, including: copy-on-write message passing, copy-on-reference network communication and extensible memory objects,
- hooks for transparent shared libraries, to provide binary compatibility with existing operating system environments.

The Mach kernel provides software equivalents of the key elements of uniprocessor and multiprocessor architectures. The Mach thread mechanism, for example, is a kind of software processor. By allowing multiple threads to run within the same program, Mach permits a system or application programmer to directly manage multiple CPUs in a multiprocessor. Mach's interprocess communication facility (IPC) provides the kind of I/O channel between threads that may exist in a multiprocessor with a message-passing bus or between workstations on a network.

Interprocess communication and memory management in Mach are tightly integrated. Memory management techniques (such as the use of memory re-mapping to avoid data copying) 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.

One of the most unusual and important facilities Mach provides is the notion of a memory object which an application program may create and manage. The memory object is like a file or data container which can be mapped into the address space of a program. Unlike traditional systems in which the operating system has complete control of "paging" data to and from such a data object, Mach allows the application which creates the memory object to act as though it were the disk storage or "pager" for that object. Mach virtual memory objects are represented as communication channels. On a page fault, the kernel sends a message to the backing storage communication channel of a memory object to get the data contained in the faulted page. This provides the flexibility necessary to implement efficiently such system applications as file systems, databases, dynamic encryption or compression of data on access or even network shared memory.

Finally, the Mach transparent library facility allows a code library to be loaded into 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. This 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 BSD, UNIX V.4),
- debugging and monitoring and
- network redirection of OS traps.

4. In-kernel OS Emulation

In the first implementation of a 4.3 BSD emulation on top of Mach, the Mach kernel is used as the lower layer of a two-tier operating system implementation. In such a scheme the Mach kernel provides support for key functions such as virtual memory, scheduling, interprocess communication and device access. The target operating system can then be implemented using these functions. In this approach the entire system, kernel and OS environment, is packaged as a unit and run in a privileged state just as in a traditional OS design. In many respects it continues to resemble the more traditional operating systems it replaces. One advantage to this approach, however, is that more than one OS environment can be implemented using the same kernel interface -- reducing the software effort required to bring a new architecture to market with several supported operating systems. Another advantage is that the basic kernel could be made freely available to all without compromising the proprietary added value of the particular operating system environment layered above it. This approach would allow companies to share the costs of porting the kernel to a new architecture.

Commercial versions of Mach available today are, in fact, examples of 4.3 BSD UNIX layered above Mach kernel primitives and packaged together with the Mach code. Although one might assume that this layered approach to UNIX implementation would be a performance disadvantage, measurements of Mach versus traditional UNIX implementations indicate otherwise. Simple compilation benchmarks on SUN 3/60 workstations, for example, run nearly 40% faster under Mach than they do under Sun Microsystems own SunOS 4.0 version of UNIX. Times for UNIX "fork" and "exec" operations are also nearly a factor of two faster under Mach than SunOS.

5. Out-of-kernel OS Emulation: Two Approaches

A second approach to building a layered operating system environment has even greater potential for open system development. The kernel can be packaged by itself as a "pure" kernel with no operating system environment. In this approach, only the kernel runs in privileged state. The rest of the operating system environment runs, in effect, as one or more programs (or, more precisely, one or more server processes) on top of the kernel. User applications run as before, but instead of making direct calls on the operating system via system calls traps, the kernel's communication and memory management facilities are employed to communicate information between the application and operating system processes. The reason this implementation strategy is so attractive for open systems, is that it can allow more than one operating system environment to be supported on the same machine, on the same kernel, at the same time. Systems such as UNIX or OS/2 could potentially co-exist in their native form. The kernel becomes a kind of universal "socket" into which more than one operating system environment can be plugged, insulating that software from the hardware itself and greatly simplifying its design and maintenance.

This approach is currently being put to the test at Carnegie Mellon in the development of two rather different user-state implementations of Berkeley UNIX 4.3 BSD: the *Multithreaded System* and the *Multiserver System*. Both implementations run unmodified 4.3 BSD binaries.

6. The Multithreaded UNIX Server

This system consists of transparent library support augmented by a multithreaded UNIX server. This server, contained in a single task, is typically invoked via a Mach message exchange for each system call issued by application processes. In addition to managing system call emulation for Unix processes, the Unix server acts as an external pager for Unix inodes. It is implemented using Mach's C-Threads package with each incoming request handled by a cthread allocated from a pool of waiting threads.

A single task operating system emulation of this kind is attractive for several reasons:

- The server is solely responsible for performing the emulation of all OS environment semantics. The structure of the server is, in fact, similar to that of an in-kernel implementation; it has global knowledge of all the information needed for the emulation. Internal context switching between threads can be extremely fast.

- The OS server is completely pageable and can in fact make more efficient use of memory (by sharing data structures and stack space) than can a multiple server implementation.

- It can be relatively straightforward to transform an existing in-kernel OS implementation into such a server, because most of the code can be simply carried over. This can make it easy to preserve both existing code and semantics. This could allow vendors with proprietary OS environments to more quickly take advantage of Mach as a basis for their systems.

In practice, this single task Unix server works well and demonstrates the feasibility of such an approach. Its implementation was completed in less than sixth months, and can already be used for self-development. It currently runs on VAX and Sun 3 platforms and is functionally interchangeable with existing versions of 4.3 BSD/Mach on those machines. We expect to extend this implementation to the other hardware platforms which run Mach and to put this version into production use within CMU over the next few months. Initial performance measurements are encouraging. A compilation benchmark which takes takes 29 seconds to complete on a Sun 3/60 running in-kernel 4.3BSD/Mach takes 34 seconds with out-of-kernel BSD support and 49 seconds running under SunOS 4.0.

7. Multiserver UNIX

This system divides responsibility for UNIX support among a collection of libraries and servers responsible for particular OS functions such as naming, authentication and file data access. Wherever possible, the interfaces between the various system components, and those components themselves, are designed to be independent of the target environment. This approach presents two major advantages:

- access to various system resources can be shared by multiple independent operating system environments, communicating over a network or concurrently executing on the same machine.

- individual components can easily be reused for the implementation of different operating system environments.

On the negative side, this approach requires sophisticated synchronization between servers to achieve precise UNIX semantics, and very careful design of the standardized interfaces. The major interfaces defined for that system organization are:

- a standard access protocol defining the authentication, access control and naming procedures used for access to all system objects such as files, devices, processes, etc.

- a standard I/O protocol for the transfer of data between the producers and consumers of that data.

- a standard exception protocol to handle exceptions happening during client-server interactions, and to report asynchronous events to clients.

The following sections describe the major components of this system.

7.1. Mach Object Programming Facility

The development of the multiserver UNIX system is aided by a C-based object-oriented programming package called *MachObjects*, 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.

To simplify the organization of the various components, libraries of standard MachObjects classes are used, that implement the standard system interfaces. In many cases, a special MachObject mechanism is used to allow a server to dynamically select the class of an object to be instantiated in its client's address space. When this approach is used, only the client-side object must implement the standard interfaces. Each server may use a different, specialized protocol to communicate with the client-side objects that it returns.

7.2. Transparent Library

The transparent library is responsible for translating the UNIX 4.3 BSD system calls from an application process into invocations on the appropriate system servers, via the corresponding MachObjects elements.

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

7.3. System Servers

The various system servers cooperate to implement the functions needed for the emulation. As indicated, many of these servers are in fact independent of the specific UNIX environment, and only the fact that they are invoked from the 4.3 BSD transparent library produces 4.3 BSD semantics from the point of view of the application processes. The major servers are or will be:

- the Name Server, implementing a hierarchical name space with only directories, symbolic links and mount points, which can be used to tie together several other name spaces and represent the "root" of the UNIX name space,
- the Task Manager, responsible for keeping track of all the application processes participating in an operating system environment,
- the Authentication Server, used to verify the credentials of processes performing operations on behalf of the authorized users of the system,
- the Network Server, implementing the transport protocols used for network access (TCP/IP, OSI, etc),
- the UNIX File Server, which manages UNIX file systems on permanent storage, but uses the standard naming and I/O protocols, so that it is accessible from all environments,
- the NFS Server, which translates requests from the standard access and I/O interface into the NFS protocol, allowing access to remote NFS file systems,
- the UNIX TTY Server, a front-end for access to terminal lines and pseudo terminals, implementing the line disciplines and
- the UNIX Pipe Server, implementing traditional UNIX pipes, using FIFO buffers in shared memory.

8. Related Work

Several other research groups are also investigating the issues involved with OS emulation, particularly with respect to the UNIX environment. CMU's Accent operating system [6] was used as the base for a System III Unix emulation called *QNIX*. The *Amoeba* system [8] uses port capabilities in a manner very similar to that of Mach to implement a fast server-based system. The *CHORUS* system [2] has adopted an object-oriented approach to build a complete UNIX emulation; its use of memory protection is however rather different from that adopted for the Mach OS emulators. The *Taos* system [4] provides an emulation of Ultrix in the Topaz environment. Several of the concepts used for the Mach OS emulators are inspired from ideas presented with the *Sprite* and *V* systems [5] [11].

9. Mach availability

The portability of Mach has been demonstrated by the range of uniprocessor and multiprocessor systems on which it is available. Mach has been ported to the VAX architecture uniprocessors and multiprocessors, the SUN 3 family, the IBM RT PC family, the DecStation 3100, the 64-processor IBM RP3, the 8-processor IBM ACE multiprocessor workstation, the Sequent Balance, the Macintosh II, the IBM 370, the SUN 4, the Intel 386 and the Intel i860. Implementations for other MIPS R2000 and R3000 machines are nearing completion and several implementations for the Motorola 88000 are underway. Commercial versions of Mach are available from BBN Advanced Computers, Evans and Sutherland Computer Division, Encore Computers and NeXT. In addition to these vendor releases of Mach, Mt Xinu, Inc. has announced that it will develop commercial end-user releases of Mach for a variety of machine architectures. Finally, Prime, Intel, Olivetti, Convergent and AT&T have recently announced a joint research project to build a multiprocessor version of System V.4 using the Mach kernel.

All software implemented by the Mach project is licensed and distributed to universities, research laboratories and corporations at no cost by Carnegie Mellon.

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. Francois Armand and Michel Gien and Marc Guillemont and Pierre Leonard. Towards a Distributed UNIX System - The CHORUS Approach. Proceedings of the European UNIX Systems User Group Conference, September, 1986.

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

4. Paul R. McJones and Garret F. Swart. Evolving the UNIX System Interface to Support Multithreaded Programs. Research Report 21, DEC Systems Research Center, September, 1987.

5. John K. Ousterhout and Andrew R. Cherenson and Frederick Douglis and Michael N. Nelson and Brent B. Welch. "The Sprite Network Operating System". *IEEE Computer 21*, 2 (February 1988), 23-36.

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

7. 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.

8. Robbert van Renesse and Hans van Staveren and Andrew S. Tanenbaum. "Performance of the World's Fastest Distributed Operating System". *ACM Operating Systems Review 22*, 4 (October 1988), 23-34.

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

10. 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.

11. Willy Zwaenepoel. "Implementation and Performance of Pipes in the V-System". *IEEE Trans. on Comput. C-34*, 12 (December 1985), 99-106.

Table of Contents