Mach Bibliography

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891

27 October 1994

This bibliography suggests background reading about the Mach operating system.

We have tried to include enough information about unpublished works so that you will be able to obtain copies without too much trouble. For all the documents that are available via anonymous FTP, the file name relative to the Mach public FTP area is given. These files can be FTP'ed from **mach.cs.cmu.edu** as user **anonymous**.

PLEASE NOTE: If you are interested in obtain a copy of a Mach thesis but can't FTP, please send mail to reports@cs.cmu.edu to request a printed copy. Doucments that are not theses and are not on-line can be obtained by sending mail to mach@cs.cmu.edu.

1. Robert V. Baron, David Black, William Bolosky, Jonathan Chew, Richard P. Draves, David B. Golub, Richard F. Rashid, Avadis Tevanian, Jr., and Michael Wayne Young.
   Mach Kernel Interface Manual
   August, 1990. Unpublished manuscript from the School of Computer Science, Carnegie Mellon University.
   File: **doc/unpublished/manual.ps**

   This manual describes the interface to the Mach kernel in detail for a wide variety of uniprocessor and multiprocessor architectures.

2. Joseph S. Barrera III.
   A Fast Mach Network IPC Implementation
   Proceedings of the Usenix Mach Symposium, November 1991.
   File: **doc/published/ipc2.ps**

   This paper describes the implementation of network Mach IPC optimized for clusters of processors connected by a fast network, such as workstations connected by an Ethernet or processors in a non-shared memory multiprocessor. This work contrasts with earlier work, such as the net msg server, which has emphasized connectivity (by using robust and widely available protocols such as TCP/IP) and configurability (with an entirely user-state implementation) at the expense of performance.

   The issues addressed by this work are support for low latency delivery of small and large messages, support for port capabilities and reference counting, and integration with the existing local Mach IPC implementation. Low latency for small messages requires careful buffer and control flow management; this work is compared with other fast RPC work described in the literature. Low latency for large messages, particularly for faster networks, requires an avoidance of copying, which can be achieved through virtual memory support; the modifications that were necessary to make Mach's virtual memory support inexpensive enough to be useful for this purpose is described. The distributed implementation of port capabilities, port reference counts, and port migration is discussed, and compared with that in the netmsg server. Finally, performance data is presented to quantify the speedup achieved with the described implementation.

3. David L. Black.
   The Mach cpu_server: An Implementation of Processor Allocation
   August 1989. Unpublished manuscript from the School of Computer Science, Carnegie

Mellon University.
File: **doc/unpublished/cpuserver.{doc,ps}**

This document describes the cpu_server, a user-mode server which performs processor allocation for the Mach operating system.

4. David. L. Black.
Scheduling Support for Concurrency and Parallelism in the Mach Operating System.
CMU Technical Report CMU-CS-90-125, April 1990. A revised version can be found in the May 1990 issues of IEEE Computer.
File: **doc/unpublished/sched.concur.{ps,dvi}**

Changes in the use of multiprocessors are placing new demands on operating system schedulers. This article describes some of the new challenges posed by parallel and concurrent applications, and introduces techniques developed by the Mach project to meet these challenges. An overview of the technique of timesharing scheduling and a description of Mach are also included.

This article describes work to incorporate processor allocation and control into the Mach operating system. The design approach divides the implementation into three components: (1)basic mechanisms implemented in the kernel, (2)long term policy implemented in a server, and (3)optional user implementation of short term policy. Isolating long term policy in a server yields the advantage of policy-mechanism separation while avoiding the drawbacks encountered in previous applications of this principle to multiprocessor scheduling. The design and implementation of a processor allocation server for a gang scheduling policy is also described.

This article also describes work to support the effective multiprogrammed use of multiprocessors. The approach taken to this problem implements the scheduler in the kernel, but encourages users to provide hints. This allows the scheduler to take advantage of user knowledge without requiring users to implement sophisticated scheduling modules.

5. David. L. Black.
Scheduling and Resource Management Techniques for Multiprocessors.
CMU Thesis CMU-CS-90-152, July 1990.
File: **doc/published/schedsupport.ps**

This thesis describes research in the design and implementation of multiprocessor resource management and related areas. Application requirements motivated the major research areas, processor scheduling and non-uniform memory management, as these areas contain the most important problems raised by the changing design and use of multiprocessors.

6. David L. Black, David B. Golub, Richard F. Rashid, Avadis Tevanian, Jr., and Michael W. Young.
The Mach Exception Handling Facility
Technical Report CMU-CS-88-129, School of Computer Science, Carnegie Mellon University, April 1988.
File: **doc/unpublished/exception.ps**

Exceptions are caused by the occurrence of unusual conditions during program execution; raising an exception invokes the operating system to manage recovery from the unusual condition. Although many exceptions can be handled and dismissed transparently to the user (e.g., page faults), some must be exported in some form (invoke handler, core dump, etc.). This paper concerns the design and implementation of exception handling facilities that perform this exporting for general purpose operating systems. It discusses both the overall design principles for these facilities and the specific design and implementation of an exception handling facility for the Mach operating system. It also discusses the support provided to applications that use exception (e.g., debuggers) by this facility and other Mach kernel facilities.

7. Brian N. Bershad, Matthew J. Zekauskas.
Shared Memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors.
CMU Technical Report CMU-CS-91-170, September 1991.
File: **doc/unpublished/midwaytr.ps**

Distributed memory multiprocessing offers a cost-effective and scalable for a large class of scientific and numeric applications. Unfortunately, the performance of current distributed memory programming environments suffers because the frequency of communication between processors can exceed that required ensure a correctly functioning program.

MW is a shard memory parallel programming system which addresses the problem of excessive communication in a distributed memory multiprocessor. MW programs are written using a conventional MIMD-style programming model executing within a single globally shared local memory. Local memories on each processor cache recently used a data to counter the effects of network latency.

MW is based on a new model of memory consistency called *entry consistency*. Entry consistency exploits the relationship between synchronization objects and the data they protect. Updates to shared data are communicated between processors only when not doing so would result in an inconsistent state given a properly synchronizing program.

MW implements its consistency protocols in software and has no dependencies on any specific hardware characteristic other than the ability to send messages between processors. A strictly software solution is attractive because it allows us to exploit application specific information at the lowest levels of the system, and because it ensures portability across a wide range of multiprocessor architectures.

PLEASE NOTE: The authors suggest that people interested in this topic should first obtain the paper: The Midway Distributed Shared Memory System. CMU Technical Report CMU-CS 93-119. The abstract is listed in this bibliography.

8. Brian Bershad.
Practical Considerations for Non-Blocking Concurrent Objects.
CMU Technical Report CMU-CS-91-183, October 1991.
File: **doc/published/non-blocking.ps**

An important class of concurrent objects are those that are *non- blocking*, this is, whose operations are not contained within a mutually exclusive critical sections. A non-blocking object can be accessed by many threads at a time, yet update protocols based on atomic cas operations guarantee the object’s consistency.

In this paper, we take a practical look at the cas operation in the context of contemporary shared memory multiprocessors. We first describe an operating system-based solution which permits the construction of a non-blocking function on architectures which only support more primitive atomic primitives such as tas or Atomic Exchange. We then evaluate several locking strategies which are used to synthesize a cas operation, and show that common techniques for reducing synchronization overhead in the presence of contention are inappropriate when used as the basis for non-blocking synchronization. We describe a simple synchronization strategy which avoids much of the overhead normally associated with contention.

9. Brian N. Bershad.
The Increasing Irrelevance of IPC Performance for Microkernel-Based Operating Systems.
Proceedings of the 1992 USENIX Workshops on Microkernels.
File: **doc/published/IPCperf.{doc,ps}**

IPC is the glue with which the traditional operating system services such as networking, and filing, are provided in microkernel-based operating systems. Because applications rely heavily on cross-address space communication, IPC performance is often viewed as being the "Achilles heel" of a microkernel-based operating system.

In this paper I discuss four reasons why the IPC performance is becoming increasingly irrelevant to overall system performance.

10. Brian N. Bershad, Richard P. Draves, Alessandro Forin.
Using Microbenchmarks to Evaluate System Performance.
Proceedings of the Third Workshop on Workstation Operating Systems (WWOS-3), April 1992.
File: **doc/published/benchmark{.doc,.ps}**

It has become nearly impossible to write a paper about anything in operating systems without including some discussion of performance. Usually, the performance sections concentrates on a few "microbenchmarks" which demonstrate that whatever it is which is being described in the paper can, has been, or might be efficiently implemented. For example, the time to execute a null remote procedure call, a system trap, or to access a page of a mapped file have all been used at one time or another to show that the system implementing the function is either efficient(if it was built by the authors)or inefficient(if it was built by anybody else).

Two implicit assumptions underlie the use of microbenchmarks. First, it is assumed that the time required for the microbenchmark to exercise the code path in question is the same as it is when the code path is used by relay programs. Second, there is the assumption that a microbenchmark is actually representative of something which is either important in its own right, or which has a measurable impact on over system performance. In this paper we point out the vulnerability of the first assumption by showing the significant variation that can occur with even the simple microbenchmark. Identifying weaknesses in the second assumption is something best done on a case by case basis.

11. Brian N. Bershad, David D. Redell*, John R. Ellis*.
Fast Mutual Exclusion for Uniprocessors.
A version of this paper will appear in the Fifth Symposium on Architectural Supports for Programming Languages and Operating Systems (ASPLOS V), October 1992
File: **doc/published/Rcs.ps**

In this paper we describe restartable atomic sequences, an *optimistic* mechanism for implementing simple atomic operations (such as *Test-And- Set*) on a uniprocessor. A thread that is suspended within a restartable atomic sequence is resumed by the operating system at teh beginning of the sequence, rather than at the point of suspension. This guarantees that the thread eventually executes the sequence *atomically*. A restartable atomic sequence has significantly less overhead than other software-based synchronization mechanisms, such as kernel emulation or software reservation. Consequently, it is an attractive alternative for use on uniprocessors that do not support atomic operations. Even on processors that do support atomic operations in hardware, restartable atomic sequences can have lower overhead.

We describe different implementation of restartable atomic sequences for the Mach 3.0 and Taos operating systems. These systems' thread management packages rely on atomic operations to implement higher-level mutual exclusion facilties. We show that improving the performance of low-level atomic operations, and therefore mutual exclusion mechanisms, improves application perfomance.

*David D. Redell and John R. Ellis are currently with Digital Equipment Corporation, Palo Alto, CA.

12. Brian N. Bershad, Matthew J. Zekauskas and Wayne A. Sawdon.
The Midway Distributed Shared Memory System.
CMU Technical Report CMU-CS 93-119. A version of the paper can also be found in the Proceedings of COMPCON 1993.
File: **doc/published/midway.ps**

This paper describes the motivation, design and performance of Midway, a programming system for a distributed shared memory multicomputer (DSM) such an ATM based cluster, a CM-5, or a Paragon. Midway supports a new memory consistency model called entry consistency. Entry consistency guarantees that shared data becomes consistent at a processor when the processor acquires a synchronization object known to guard the data. Entry consistency is weaker than other models described in the literature, such as processor consistency and release consistency, but it makes possible higher performance implementations of the underlying consistency protocols. Midway programs are written in C, and the association between the synchronization objects and data must be made with explicit annotations. As a result, pure entry consistent programs can require annotations than programs written to other models. In addition to entry consistency, Midway also supports the stronger release consistent and processor consistent models at the granularity of individual data items. Consequently, the programmer can tradeoff potentially reduced performance for the additional programming complexity required to write an entry consistent parallel program.

13. Jose C. Brustoloni and Brian N. Bershad.
    Simple Protocol Processing for High-Bandwith Low-Latency Networking
    Technical Report CMU- CS-93-132, School of Computer Science, Carnegie Mellon University, March 1992.
    File: **doc/unpublished/atm.{ps,dvi}**

    Protocol and operating system overheads have become the limiting factor for communications performance on fast networks such as ATM. A large component of these overheads stems fromm the *protocol redundancy* that arises when layering higher level protocols on top of lower level ones. ATM, for example, requires specific mechanisms for connection management, flow control, congestion avoidance, and segmentation and reassembly. With these mechanisms in place, it is relatively simple and inexpensive to provide for reliable sequenced delivery at the network interface level, making similar functionality in higher-level protocols such as TCP/IP redundant. In this paper we present a protocol architecture specifically tailored for communication over high-bandwith low-latency local or metropolitan ATM networks. Our architecture yields high performance by eliminating protocol redundancy and by exploiting common-case communication behavior. With this approach, we can combine the functionality typically found in four separate layers of the ISO model - data like through session - in a single pass over the data, delivering high throughput and low latency. Our protocol architecture requires minimal hardware support from the network interface and switch fabric, yet effciently provides services such as segmentation and reassembly, flow control, congestion avoidance, and error recovery. We have implemented our protocol architecture on a switch-based ATM network consisting of DECstation 5000/200 workstations running the Mach 3.0 operating system. Our implementation achieves latencies and bandwiths close to the physical limitations imposed by the hardware, yet offers applications a high-level reliable transport interface.

14. Deborah Casewell, David. Black.
    Implementing a Mach Debugger for Multithreaded Applications
    CMU Technical Report CMU-CS-89-154, November 1989. Also can be found in the Proceedings of Winter 1990 USENIX Technical Conference and Exhibition, January 1990.
    File: **doc/published/debugger.ps**

    Multiple threads of control add new challenges to the task of application debugging and require the development of new debuggers to meet these challenges. This paper describes the design and implementation of modifications to an existing debugger (gdb) for debugging multithreaded applications under the Mach operating system. It also describes the operating facilities that support it. Although certain implementations details are specific to Mach, the underlying design principles are applicable to other systems that support threads in a Unix compatible environment.

15. J. Bradley Chen, Brian N. Bershad.
    The Impact of Operating System Structure on Memory System Performance.

To appear in the proceedings of The Fourteenth Symposium on Operating System Principles.
File: **doc/published/os-memorysys.ps**

In this paper, we evaluate the memory system behavior of two distinctly different implementations of the UNIX operating system: DEC's Ultrix, a monolithic system, and Mach 3.0 with CMU's Unix server, a microkernel-based system. In our evaluation we use combined system and user memory reference traces of thirteen industry-standard workloads. We show that the microkernel-based system executes substantially more non-idle system instructions for an equivalent workload than the monolithic system. Furthermore, the average instruction for programs running on Mach has a higher cost, in terms of memory cycles per instruction, than on Ultrix. In the context of our traces, we explore a number of popular assertions about the memory system behavior of modern operating systems, paying special attention to the effect that Mach's microkernel architecture has on system performance. Our results indicate that many, but not all of the assertions are true, and that a few, while true, have only negligible impact on real system performance.

16. J. Bradley Chen.
Software Methods for System Address Tracing.
Technical Report CMU-CS-93-188, School of Computer Science, Carnegie Mellon University, August 1993. Also, to appear in the proceedings of The Fourth Workshop on Workstation Operating Systems, Napa, CA, October 14-15.
File: **doc/published/system.address.tracing.ps**
File: **doc/unpublished/system.address.tracing.ps**

This paper discusses the use of software methods to collect system trace for DEC Ultrix and 3.0 Mach on a DECstation 5000/200. We assert that software methods are a valuable tool for collecting system trace and understanding operating system and memory system behavior for modern workstation workloads. Software methods have some well documented shortcomings. We discuss how their impact was minimized in our system. We further support the validity of the software support approach by comparing behavior predicted by our tracing/simulation system to measurements made with less intrusive methods.

17. J. Bradley Chen.
Memory Behavior of an X11 Window System.
This paper also appears in the Proceedings of the Winter USENIX Conference, January 1994.
File: **doc/published/X11_mem_behavior.ps**

We used memory reference traces from a DEC Ultrix system running the X11 window system from MIT Project Athena and several freely available X11 applications to measure different aspects of memory system behavior and performance. Our measurements show that memory behavior for X11 workloads differs in several important ways from workloads more traditionally used in cache performance studies. User instruction cache behavior is a major component in overall memory system delays, with significant competition within and between address spaces. User TLB miss rates are up to a factor of two higher than other ill-behaved integer workloads. Write-buffer stalls, data cache behavior, and uncached memory reads can be problematic for microbenchmarks, but they are not an issue for the realistic applications we tested.

18. J. Bradley Chen.
The Impact of Software Structure and Policy on CPU and Memory System Performance
Carnegie Mellon Thesis: CMU-CS-94-145
File: **1994/CMU-CS-94-145.ps.Z on the machine reports.adm.cs.cmu.edu**

Operating systems, when compared to application programs, have received disappointingly little benefit from the performance improvements of the most recent generation of microprocessors. This thesis used complete traces of software activity from a RISC-based uniprocessor to expose the dynamic behavior of operating system execution and explore the sources of poor performance. Traces from both Mach 3.0 and Ultrix implementations of UNIX permitted a study of performance differences between microkernel and monolithic implementations of the same operating system interface. The comparison showed that both system structure and policy implemented in the system have a significant impact on performance.

Measurements of X11 workloads showed that memory system behavior for these large workloads differs significantly from the kinds of workloads traditionally used for performance analysis. Structural and behavioral similarities between large X11 workloads and the operating system are reflected in their overall performance.

These experiments represent the first application of address tracing with software instrumentation to the study of behavior in a popular operating system. The development of the tools and their application in this research demonstrates that such methods are a powerful tool for understanding behavior and interactions in complex software systems. Overall, the experiments demonstrate that system level effects must be measured for a complete understanding of overall performance.

19. Eric C. Cooper and Richard P. Draves.
    C Threads.
    Technical Report CMU- CS-88-154, School of Computer Science, Carnegie Mellon University, February 1988.
    File: **doc/unpublished/threads.ps**

    The C Threads package allows parallel programming in C under the Mach operating system. The package provides multiple threads of control within a single shared address space, mutual exclusion locks for protection of critical regions, and condition variables for thread synchronization.

20. Randall Dean, Francois Armand*.
    Data Movement in Kernelized Systems
    File: **doc/unpublished/datamovement.{ps, doc}**

    This paper considers how two kernelized systems, Mach 3.0 with the BSD4.3 Single Server and Chorus/MiX V.4, move data to and from files under a variety of circumstances. We give an overview of the kernel abstractions and system server and describe in detail the *read*() and *write*() paths of these two systems. We then break down their *read*() and *write*() performance and compare them to two monolithic systems, Mach 2.6MSD(BSD4.3) and System V R4.0. We then describe the compromises each of the 2 kernelized systems make in order to achieve a goal of performance comparable to the monolithic systems. We conclude with a description of what techniques each system uses that could benefit both each other and traditional monolithic systems.

    *Francois Armand is currently with CHORUS Systems-Cedex, France.

21. Randall Dean.
    Using Continuations to Build a User-Level Threads Library
    This paper will appear in the Proceedings of the Third USENIX Mach Conference, April 1993.
    File: **doc/published/cont_threads.ps**

    We have designed and built a user-level threads library that uses *continuations* for transfers of control. The use of *contiuations* reduces the amount of state that needs to be saved and restored at context switch time, thereby reducing the instruction count in the critical sections. Our multiprocessor contention benchmarks indicate that this reduction and use of *Busy Spinning, Busy Waiting and Spin Polling* increases throughput by as much as 75% on a multiprocessor. In addition, flattening the locing hierarchy reduces context switch latency

by 5% to 49% on both uniprocessors and multiprocessors. This paper describes the library's design and compares its overall performance characteristics to the existing implementations.

22. Richard P. Draves.
A Revised IPC Interface.
Proceedings of the USENIX Mach Conference, October 1990.
File: **doc/published/ipc.ps**

The Mach 3.0 IPC facility efficiently supports many different styles of communication, including server-client remote procedure calls, distributed object-oriented programming, and streams. As a building block, the interface provides a general message primitive. The message primitive operates on capabilities for communication ports. The interface remedies numerous problems that were observed in the older Mach 2.5 interface. The implementation provides good performance, running 30%-40% faster than Mach 2.5 with data structures 50% smaller. The Mach 3.0 implementation provides backwards-compatibilities for the older interface.

23. Richard P. Draves, Michael B. Jones, and Mary R. Thompson.
MIG --- The MACH Interface Generator.
July 1989. Unpublished manuscript from the School of Computer Science, Carnegie Mellon University.
File: **doc/unpublished/mig.ps**

Matchmaker is a language for specifying and automating the generation of multilingual interprocess communication interfaces. MIG is an interim implementation of a subset of the Matchmaker language that generates C and C++ remote procedure call interfaces for interprocess communication between Mach tasks.

24. Richard P. Draves, Brian N. Bershad, Richard F. Rashid, Randall W. Dean.
Using Continuations to Implement Thread Management and Communication in Operating Systems.
Technical Report CMU-CS-91-115, Carnegie Mellon University, October 1991. Also appears in the proceedings of the Thirteenth Symposium on Operating Systems(SOSP).
File: **doc/published/threadmgnt.{dvi,ps}**

We have improved the performance of the Mach 3.0 operating system by redesigning its internal thread and interprocess communication facilities to use *continuations* as the basis for control transfer. Compared to previous versions of Mach 3.0, our new system consumes 85% less space per thread. Cross-address space remote procedure calls execute 14% faster. Exception handling runs over 60% faster.

In addition to improving system performance, we have used continuations to generalize many control transfer optimizations that are common to operating systems, and have recast those optimizations in terms of a single implementation methodology. This paper describes our experiences with using continuations in the Mach operating system.

25. Richard P. Draves.
Page Replacement and Reference Bit Emulation in Mach.
Proceedings of the Usenix Mach Symposium, November 1991.
File: **doc/ublished/pagereplace.{dvi,ps}**

A page-replacement algorithm guides the operating system when it needs to create more free pages by reclaiming pages that are filled with data. The Mach kernel approximates the familiar Least Recently Used (LRU) algorithm with an algorithm known as FIFO with Second Chance. This Strategy uses page-referenced information to avoid reclaiming recently-referenced pages. On hardware lacking such support, the kernel must detect references in software.

This paper describes the Mach kernel's page-replacement algorithm and considers three software techniques for detecting references. It shows that Mach 3.0's reference fault

technique output outperforms Mach 2.5's reactivation fault technique, and also outperforms reference detection via a software TLB miss handler. Based on performance of the Mach 3.0 implementation, we conclude that hardware page-referenced information is not a prerequisite for a satisfactory page-replacement algorithm.

26. Alessandro Forin, Joseph Barrera, Michael Young, Richard Rashid.
Design, Implementation and Performance Evaluation of a Distributed Shared Memory Server for Mach.
A shorter version of this paper appears in the Proceedings of the Winter USENIX Conference, January 1989.
File: **doc/unpublished/netmemorysrv.{doc,ps}**

This report describes the design, implementation and performance evaluation of a virtual shared memory server for the Mach operating system. The server provided unrestricted sharing of read-write memory between tasks running on either strongly coupled or loosely coupled architectures, and any mixture thereof. A number of memory coherency algorithms have been implemented and evaluated, including a new distributed algorithm that is shown to outperform centralized ones. Some of the features of the server include support for machines with multiple page sizes, for heterogeneous shared memory, and for fault tolerance. Extensive performance measures of applications are presented, and the intrinsic costed evaluated.

27. Alessandro Forin, David Golub, Brian Bershad.
An I/O System for Mach.
Proceedings of the Usenix Mach Symposium, November 1991.
File: **doc/published/IO.{dvi,.ps}**

The Mach 3.0 I/O system represents a radical departure from its predecessor-Mach 2.5, which relied on the BSD Unix model of device management. The I/O interface in Mach 3.0 supports device drivers that are largely device-independent, implemented at user-level, and location-independent. Our approach to device management significantly reduces the size of the kernel's machine-dependent code, enables us to reduce the length of the I/O path, and permits us transparently manage remote devices on non-shared memory multiprocessor architectures such as the Hypercube. This paper describes the structure and performance of Mach's I/O system.

28. Alessandro Forin and Gerald R. Malan.
An MS-DOS File System for UNIX.
An extended version of a paper that appeared in the 1994 Winter USENIX Conference.
File: **doc/published/dos-fs.ps**

We have written DosFs, a new file system for UNIX that uses MS-DOS data structures for permanent storage. DosFs can be used anywhere a traditional UNIX file system can be used, and it can mount disks written by MS-DOS as regular UNIX partitions. DosFs can be used as the root partition, and exported by an NFS server. Our motivation for this work was efficient disk space utilization; DosFs provides from 10% to 13% better disk utilization than the 4.3 BSD Fast File System (FFS). In addition, we found that the disk block allocation algorithm used by DosFs lets us exploit large contiguous disk operations, providing a five-fold improvement over FFS for uncached operations. To the best of our knowledge, this is the first file system implementation that allows the user to specify the logical block size at mount time. A user can mount the same file system with a larger block size when file accesses tend to be sequential and a smaller one when they tend to be scattered. The MS-DOS structures were designed for a single-user system and do not support access control. We solved this problem with simple extensions that are backward-compatible with MS-DOS.

29. Michael Ginsberg, Robert V. Baron, Brian N. Bershad.
Using the Mach Communication Primatives in X11.
Proceedings of the Third USENIX Mach Conference, April 1993. The paper will soon be

available as technical report: CMU-CS-93-121.
File: **doc/published/mach-in-x11.ps**

We have modified the X11 windowing system to use the native communication facilites of the Mach 3.0 microkernel. Our new implementation can rely on Mach's low-overhead IPC facility as a direct replacement for sockets, or it can use shared memory as a transport between X11 clients and the server. On conventional BSD Unix systems, X11 communication is done through sockets. Because a user-level process inplements Unix functionality on top of Mach 3.0, a socket-based version of X11 performs substantially worse than when running on a monolithic Unix kernel. Using Mach IPC as the transport between X11 clients and the server, X11 performance is slightly better than that of a monolithic system in which sockets are implemented inside the kernel as opposed to within a user level process. Using Mach's shared memory facilities as the transport, we have measured performance improvements of over 40%.

30. David Golub, Randall Dean, Alessandro Forin, Richard Rashid.
Unix as an Application Program.
This article appears in the Proceedings of the USENIX Summer Conference, June 1990.
File: **doc/published/mach3_intro.ps**

This is an introduction to the Mach 3.0 kernel.

Since March 1989 we have been running at CMU a computing environment in which the functions of a traditional Unix system are cleanly divided into two parts:facilities which manages the hardware resources of a computer system(such as CPU. I/O and memory)and support for higher-level resource abstractions used in the building of application programs, e.g. files and sockets. This paper describes the implementation of Unix as a multithreaded application program running on the Mach kernel. The rationale, design, implementation history and performance of the system is present.

31. David B. Golub, Richard P. Draves.
Moving the Default Memory Manager out of the Mach Kernel.
Proceedings of the Usenix Mach Symposium, November 1991.
File: **doc/published/defaultmm.ps**

We have implemented a default memory manager for the Mach 3.0 kernel that resides entirely in user space. The default memory manager uses a small set of kernel privileges to lock itself into memory, preventing deadlocks against other Mach systems services. An extension to the Mach boot sequence loads both the kernel and user program images at system startup time. The resulting system allows the default memory manager to be built and run in a standard user-level environment, but still operates with the high reliability required by the Mach kernel.

The default memory manager is bundled with another component of the Mach 3 system: the bootstrap service. This service starts the initial set of system servers that make up a complete operating system based on the Mach 3.0 kernel. Since the real file system may be one of these servers, the bootstrap service needs its own copy of a subset of the file system. This is shared with the default memory manager. Placing these two components outside the kernel allows them to be easily reconfigured with different file systems.

32. Paulo Guedes(Open Software Foundation), Daniel Julin.
Object-Oriented Interfaces in the Mach 3.0 Multi-Server System.
Proceedings of the IEEE Second International Workshop on Object Orientation in Operating Systems, October 1991.
File: **doc/published/multiserver_interface.ps**

The Mach 3.0 multi-server system decomposes the functionality of the operating system between the micro-kernel, a set of system servers running in user-mode and an emulation library executing in the address space of applications. The interfaces provided by the system servers are object -oriented and both the server and emulation library are written in an object-oriented language.

In this paper we present how the interfaces between the components are specified and implemented to guarantee consistency and early detection of errors, yet maintaining the flexibility to extend and configure the system by adding new or modified servers without affecting exiting pieces.

33. Indira Subramanian.
Managing Discardable Pages with an External Pager.
Proceedings of the Usenix Mach Symposium, November 1991.
File: **doc/published/discardable-pager.ps**

We have designed and implemented an external pager that: (1) receives information regarding discardability of pages from the client such as a garbage collector for functional programming languages, (2) saves and restores only non-discardable pages and (3) influences page-replacement by pre-flushing discardable pages. In a general purpose operating system dirty pages are typically saved to disk and then restored from disk even when the application may not care about the contents of those pages. For example, copying garbage collection used in functional programming languages generates many pages that are dirty but discardable. Using the external pager to manage the discardable pages we observed that elapsed times for some applications decreased by as much as a factor of 6.

34. Michael B. Jones, Richard F. Rashid.
Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems.
Technical Report CS-CMU-88-129, School for Computer Science, Carnegie Mellon University, September 1986.

Mach, a multiprocessing operating system kernel providing capability-based interprocess communtication and Matchmaker, a language for specifying and automating the generation of multi-ligual interprocess communication interfaces, are presented. Their usage together providing a heterogeneous, distributed, object-oriented programming environment is described. Performance and usage statistics are presented. Comparisons are made between the Mach/Matchmaker environment and other related systems. Possible future directions are examined.

35. Michael B. Jones.
Bringing the C Libraries With Us into a Multi-Thread Future.
Proceedings of the 1991 Winter USENIX Conference, January 1991.
File: **doc/published/cmultithread.{ps}**

An enormous amount of UNIX (and UNIX-like) code has been written (by a likewise enormous amount of programmers) that uses the standard C libraries. Use is made throughout much of this code of the knowledge that traditional UNIX programs have exactly one thread in control. However increasing numbers of UNIX-like systems are beginning to provide support for programs with multiple threads of control. To the possible extent, it is highly desirable to preserve the existing C library interfaces for multi-threaded programs; this will aid both in code and programmer portability between traditional UNIX environments and new multi-threaded ones.

A number of issues must be confronted in order to produce versions of the C libraries which can be used in multi-threaded programming environments. Among these are: functions with non-reentrant interfaces, functions which maintain state between invocations, use of macros in the library interfaces, interactions with signals, compatibility with single-threaded library data structures, performance issues, and of course, erno. Despite these and other problems, experience has shown that reasonable solutions are available. This paper presents both a detailed explanation of the inherent problems in producing multi-thread-safe C libraries and the different solutions which are available. Finally, the solutions to these problems adopted by a number of research and industry groups are presented.

36. Michael B. Jones.
A Toolkit for Interposing User Code at the System Interface.
IEEE Computer Society Technical Committee on Operating Systems and Application Environments(TCOS)Newsletter, Spring 1991.
File: **doc/unpublished/toolkit.{ps}**

Many contemporary operating systems utilize a system call interface between the operating systems and its clients. Increasing numbers of systems are provided low-level mechanisms for intercepting and handling systems calls in user code. Nonetheless, they typically provide no higher-level tools or abstractions for effectively utilizing these mechanisms. Using them has typically required reimplementation of a substantial portion of the system interface from scratch, making the use of such facilities unwieldy at best.

I assert that an object-oriented toolkit can be constructed which substantially increases the ease of interposing user code between clients and instances of the system interface by allowing such code to be written in terms of the high-level objects provided by this interface, rather than in terms of the intercepted systems calls themselves.

Furthermore, I assert that having such a toolkit will enable new interposition agents to be written which otherwise would not have been attempted. Examples of interesting agents include: system call tracing tools, protected environments for running untrusted binaries, modified filesystem namespaces, transparent network data compression and/or encryption agents, transactional software environments and emulators for other operating system environment.

37. Michael B. Jones.
Transparently Interposing User Code at the System Interface.
Proceedings of the Third Workshop on Workstation Operating Systems(WWOS-3), April 1992.
File: **doc/published/interpossys.ps**

Many contemporary operating systems utilize a system call interface between the operating systems and its clients. Increasing numbers of systems are providing low-level mechanisms for interception and handling system calls in user code. Nonetheless, they typically provide no higher-level tools or abstractions for effectively utilizing these mechanisms. Using them has typically required reimplementation of a substantial portion of the system interface from scratch, making the use of such facilities unwieldy at best.

I am currently constructing an object-oriented toolkit which substantially increase the ease of interposing user code between clients and instances of the system interface by allowing such code to be written in terms of the high-level objects provided by this interface, rather than in terms of the intercepted system calls themselves. This toolkit helps enable new interposition agents to be written which otherwise would not have been attempted.

I am also currently using the toolkit to construct several agents including: protected environments for running untrusted binaries, modified filesystems namespaces, logical devices implemented entirely in user space, transparent network data compression, and/or encryption agents, and system call tracing tools. Examples of other interesting agents which could be built include: transactional software environments, and emulators for other operating system environments.

Having the flexibility to simultaneously support several variant sets of system services may be particularly useful in a workstation environment: a diversity of software which expects different operating system environments can be supported on a single operating base. Even just the ability to provide multiple configurable views of the filesystem namespace provides a flexibility not found in traditional operating systems.

38. Michael B. Jones.
Inheritance in Unlikely Places: Using Objects to Build Derived Implementations of Flat Interfaces.

Proceedings of the Third International Workshop on Object Orientation in Operating Systems, September, 1992
File: **doc/published/implementation.ps**

Traditional operating systems typically provide a number of flat, non-object oriented interfaces through which system services are provided to applications. Two such examples are the Unix* system call interface and the X Window System** protocol. Typically such services are provided by large, monolithic implementations, with no provision for deriving related implementations through inheritance or other means.

Nonetheless, it is possible to provide flexible, object oriented views of otherwise flat, non-object oriented interfaces. Furthermore, these views can be used to build new implementations of these same flat interfaces in terms of other ones. This approach can bring the benefits of objects such as enhanced and/or application specific implementations of standard services to both building and using otherwise fixed, flat interfaces which for one reason or another cannot be changed.

My recent thesis work has taken this approach; other seemingly unrelated systems also uses this paradigm. This paper presents this paradigm and some of the benefits derived from it, citing several real systems as examples.

*-Unix is a trademark of UNIX Systems Laboratories, Inc. **-The X Window System is a trademark of the Massachusetts Institute of Technology.

39. Michael B. Jones.
Transparently Interposing User Code at the System Interface.
Carnegie Mellon Thesis: CMU-CS-92-170
File: **1992/CMU-CS-92-170.ps.Z on the machine reports.adm.cs.cmu.edu**

Many contemporary operating systems utilize a system call interface between the operating system and its clients. Increasing numbers of systems are providing low-level mechanisms for intercepting and handling systems calls in user code. Nonetheless, they typically provide no higher-level tools or abstractions for effectively utilizing these mechanisms. Using them has typically required reimplementation of a susbstantial portion of the system interface from scratch, making use of such facilities unwieldy at best.

This dissertation presents a toolkit that substantially increases the ease of interposing user code between clients and instances of the system interface by allowing such code to be written in terms of the high-level objects provided by this interface, rather than in terms of the intercepted system calls themselves. This toolkit helps enable new interposition agents to be written, many of which would not otherwise have been attempted.

This toolkit has also been used to construct several agents including: system call tracing tools, file reference tracing tools, and customizable filesystem views. Examples of other agents that could be built include: protected environments for running untrusted binaries, logical devices implemented entirely in user space, transparent data compression and/or encryption agents, transactional software environments, and emulators for other operating system environments.

40. Dan Julin.
Network Server Design, Mach Networking Group.
September 1989.
File: **doc/unpublished/netmsgserver.{doc,ps}**

This document describes the design of the network server and details the interfaces to the modules that are used to implement it. It is intended for implementers rather than users. This document reflects the current state of the implementation, and does not constitute a commitment for future developments.

41. Daniel P. Julin, Jonathan J. Chew, J. Mark Stevenson, Carnegie Mellon University. Paulo Guedes, Paul Neves*, Paul Roy, Open Software Foundation.

Generalized Emulation Services for Mach 3.0: Overview, Experiences and Current Status.
Proceedings of the Usenix Mach Symposium, November 1991.
File: **doc/published/mach_us-multiserver.ps**

This paper reports on an ongoing project to develop a general understanding of the problems encountered when building emulators for various operating systems at the user-level on top of a Mach 3.0 micro-kernel, and to propose a common framework to construct emulations for a wide range of target systems and environments. It presents an overview and discussion of the major techniques and experiments that characterize the design of the proposed system. Some of the relevant aspects include the combination of several independent servers to create a complete system, generic service interfaces relying on the emulation library as an interface translator, the use of object-oriented technology to define standard interfaces and to simplify the implementation of common facilities, moving portions of the system state and processing from the servers into a smart emulation library, and a few general-purpose facilities that simplify the generation of a complete system. A number if practical observations and experiences are also presented, in the context of the development of a prototype for the emulation of UNIX 4.3 BSD.

*Paul Neves is currently with the Swiss Bank Corporation/O'Connor Assoc. L.P., Chicago, IL

42. Ted Lehr, David L. Black.
Mach Kernel Monitor(with applications using the PIE environment
. February, 1990.
File: **doc/unpublished monmanual.{doc, ps}**

Factors such as the decomposition of parallel programs affect their performance. Measurements of parallel program performance are improved in supported by information such as how programs are scheduled. This manual describes how to use MKM (Mach Kernel Monitor). Special examples of data obtained by using MKM are shown via the PIE performance monitoring environment.

43. Chris Maeda, Brian N. Bershad.
Networking Performances for Microkernels.
Proceedings of the Third Workshop on Workstation Operating Systems(WWOS-3), April 1992.
File: **doc/published/netperf.ps**

Performance measurements of network protocols in microkernel systems have been discouraging, typically 2 to 5 times slower than comparable macrokernel systems. This disparity has led many to conclude that the microkernel approach, where protocols reside at user-level, in inherently flawed and that protocols must be placed in the kernel to get reasonable performance. We show that user-level network protocols have performed poorly because they rely on code designed to run in a kernel environment. As a result, they make assumptions about the costs of primitive operations such as scheduling, preemption, and data transfer which can require substantial overhead to satisfy at user level. Good user-level protocol performances can be achieved by restructuring protocol servers to take advantage of microkernel facilities, rather than ignore them.

44. Chris Maeda, Brian N. Bershad.
Protocol Service Decomposition for High-Peformance Networking.
To be presented at the 14th ACM Symposium on Operating Principles, December 1993.
File: **doc/published/user.level.protocols.ps**

In this paper we describe a new approach to implementing network protocols that enables them to have high performance and high flexibility, while retaining complete conformity to existing application programming interfaces. The key insight behind our work is that an application's interface to the network is distinct and separable from its interface to the operating system. We have separated these interfaces for two protocol implementations, TCP/IP and UDP/IP, running on the Mach 3.0 operating system and UNIX server. Specifically, library code in the application's address space implements the network protocols and transfers data to and from the network, while an operating system server manages the heavyweight abstractions that applications use when manipulating the network through operations other than send and receive. On DECstation 5000/200 systems connected by 10Mb/sec Ethernet, this approach to protocol decomposition achieves TCP/IP throughput of 1088 KB/second, which is comparable to that of a high-quality in-kernel TCP/IP implementation, and substantially better than a server-based one. Our approach achieves small-packet UDP/IP round trip latencies of 1.23 ms, again comparable to a kernel-based implementation and more than twice as fast as a server-based one.

45. Gerald Malan.
Mach DOS Update.
Open Software Foundation Newsletter, Spring 1992.
File: **doc/unpublished/mdos.ps**

Approximately a year and a half ago we began work here at Carnegie Mellon on a project to implement DOS as a user level application based on Mach 3.0 kernel. To date we support DOS as a multi-threaded Mach Task running in conjunction with either the 4.3 BSD Unix Server or with the the OSF/1 Single Server. The Mach DOS software support consists of machine dependent kernel support, and a multi-threaded emulator task which provides virtual devices, BIOS emulation, support for common DOS extensions, emulation of many DOS system calls, and software for integration with the Unix server. This work was outlines in an article presented at the Second USENIX Mach Symposium. That article appears in the proceedings of the Symposium.

46. Gerald Malan, Richard Rashid, David Golub, Robert Baron.
DOS as a Mach 3.0 Application.
Proceedings of the Usenix Mach Symposium, November 1991.
File: **doc/published/dospaper.{dvi,ps}**

We have implemented support for the DOS operating system on top of the Mach 3.0 kernel. This support included machine-dependent kernel modifications for the i386/i486 architecture to handle virtual 8086 mode, a multithreaded emulation of the IBM PC's VGA display and I/O devices, direct support for a number of common DOS functions and frequently loaded DOS drivers, and code to integrate DOS functionality with the existing 4.3 BSD Unix Server. The resulting systems allows multiple virtual DOS environments, supports DOS versions 3.1 to 5.0, and is capable of running common DOS software including performance sensitive PC entertainment software such as Wing Commander-a high speed space combat simulation system.

Many lessons were learned during the course of this work. DOS stresses a number of Mach features infrequently used by Unix emulation. The timing and latency demands of DOS applications, especially those with animation and sound, are dramatically different from those of typical Uni\x applications. Because most DOS programs interact intimately with the underlying PC hardware, DOS emulation expanded our knowledge about the behavior of Mach 3.0 when asked to provide a virtual machine environment rather than pure client/server support. In particular, quirks of the PC architecture-such as support for the so-called "high memory area" above 0x1000000-had to be precisely emulated.

This paper describes our implementation, its capabilities and limitations \ ,and the lessons learned about Mach in the course of our development effort.

47. Richard Rashid.
Threads of a New System.
Unix Review Vol. 4, No. 8, Pgs:37-49.

This article describes the origin of the Mach operating system.

48. R. F. Rashid.
From RIG to Accent to Mach: The Evolution of a Network Operating System.
In Proceedings of the ACM/IEEE Computer Society, Fall Joint Computer Conference, ACM, November 1986.

This paper describes experiences gained during the design, implementation and use of the CMU Accent Network Operating System, its predecessor, the University of Rochester RIG system and its successor CMU's Mach multiprocessor operating system. It outlines the major design decisions on which the Accent kernel was based, how those decisions evolved from the RIG experiences and how they had to be modified to properly handle general purpose multiprocessors in Mach. Also discussed are some the the major issues in the implementation of message-based systems, the usage patterns observed with Accent over a three year period of extensive use at CMU and a timing analysis of various Accent functions.

49. Richard Rashid, Avadis Tevanian, Jr., Michael Young, David Golub, Robert Baron, David Black, William Bolosky, and Jonathan Chew.
Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures.
Technical Report CMU-CS-87-140, School of Computer Science, Carnegie Mellon University, July 1987. Also in Proceedings of the 2nd Symposium on Architectural Support for Programming Languages and Operating Systems, ACM, October, 1987.
File: **doc/published/MIvmm.ps**

This paper describes the design and implementation of virtual memory management within the CMU Mach Operating System and the experiences gained by the Mach kernel group in porting that system to a variety of architectures. As of this writing, Mach runs on more than half a dozen uniprocessors and multiprocessors including the VAX family of uniprocessors and multiprocessors, the IBM RT PC, the SUN 3, the Encore MultiMax, the Sequent Balance 21000 and several experimental computers. Although these systems vary considerably in the kind of hardware support for memory management they provide, the machine-dependent portion of Mach virtual memory consists of a single code module and its related header file. This separation of software memory management from hardware support has been accomplished without sacrificing system performance. In addition to improving portability, it makes possible a relatively unbiased examination of the pros and cons of various hardware memory management schemes, especially as they apply to the support of multiprocessors.

50. Richard Rashid, Daniel Julin, Douglas Orr, Richard Sanzi, Robert Baron, Alessandro Forin, David Golub, Michael Jones.
Mach: A System Software kernel.
Proceedings of the 34th Computer Society International Conference COMPCON 89, February 1989.
File: **doc/published/syskernel.ps**

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 intergrated with the MACH IPC mechanisms.

51. Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, Richard Sanzi.
Mach: A Foundation for Open Systems.
Proceedings of the Second Workshop on Workstation Operating Systems(WWOS2), September 1989.
File: **doc/published/intro.ps**

This paper is an introduction to the Mach operating system.

52. Stefan Savage, Hideyuki Tokuda.
Real Time Mach: Exporting Time to the User.
This paper is scheduled to appear in the Proceedings of the Third Usenix Mach Symposium(Machnix), April 19-21, 1993
File: **doc/published/RTtimer.ps**

The current CMU Mach 3.0 microkernel exports simple timestamp and delay abstractions through `host_get_time()` and a timeout parameter to `mach_msg()`. While this is sufficient for many purposes, it does not provide the precision or generality required for a variety of real-time applications. In this paper we describe extensions to CMU's Mach 3.0 which provide users with flexible time-based synchronization and timestamp services. Additionally, we will describe how timing and scheduling services are integrated to allow real-time applications to handle *timing faults*.

53. J. Mark Stevenson, Daniel P. Julin.
Client-Server Interactions in Multi-Server Operating Systems: The Mach-US Approach
CMU Tech Report CMU-CS-94-191, September 1994. File: **doc/mach_us/us_client_server.ps** This paper describes requirements placed upon client-server interaction in a multi-server operating system and how to answer those requirements. Addressed are the problems of binding maintenance in the face of: remote method invocation, forking, binding transfer, authentication, asynchronous interruption, and client crashes.

Design and implementation choices are analyzed. The solutions, used by the CMU Mach-US multi-server UNIX 4.3BSD emulation on the Mach3.0 kernel, are described.

The lessons learned are applicable to multi-server OS design and should be applicable to object based systems that must resolve these binding problems in a micro-kernel environment.

54. J. Mark Stevenson, Daniel P. Julin.
Mach-US: Unix On Generic OS Object Servers
Proceedings of the 1995 USENIX Conference, January 1995. File: **doc/mach_us/us_analyze.ps** This paper examines the Mach-US operating system, its unique architecture, and the lessons demonstrated through its implementation.

Mach-US is an object-oriented multi-server OS which runs on the Mach3.0 kernel. Mach-US has a set of separate servers supplying orthogonal OS services and a library which is loaded into each user process. This library uses the services to generate the semantics of the Mach2.5/4.3BSD application programmers interface (API). This architecture makes Mach-US a flexible research platform and a powerful tool for developing and examining various OS service options.

We will briefly describe Mach-US, the motivations for its design choices, and its demonstrated strengths and weaknesses. We will then discuss the insights that we've acquired in the areas of multi-server architecture, OS remote method invocation, Object Oriented technology for OS implementation, API independent OS services, UNIX API re-implementation, and smart user-space API emulation libraries.

55. Avadis Tevanian, Jr.
Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments.
PhD thesis, School of Computer Science, Carnegie Mellon University, December 1987. Also as Technical Report CMU-CS-88-106 from the School of Computer Science, Carnegie Mellon University, 1987.

This thesis describes the design and implementation of a new, portable, memory management system and evaluates it by direct comparison with commercially developed

memory management systems. It is in production use at CMU, a number of industrial organizations and research laboratories, and is a central component of the Mach operating system, a distributed and parallel operating system developed at Carnegie Mellon University. At the time of this writing, it runs on more than half a dozen uniprocessors and multiprocessors including the DEC VAX family of uniprocessors and multiprocessors, the IBM RT PC, the SUN 3, the Encore MultiMax, the Sequent Balance 21000 and several experimental computers.

56. Avadis Tevanian, Jr. and Richard F. Rashid.
Mach: A Basis for Future UNIX Development.
Technical Report CMU-CS-87-139, School of Computer Science, Carnegie Mellon University, Pittsburgh, June 1987.

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.

57. Avadis Tevanian, Jr., Richard F. Rashid, David B. Golub, David L. Black, Eric Cooper, and Michael W. Young.
Mach Threads and the Unix Kernel: The Battle for Control.
Technical Report CMU-CS-87-149, School of Computer Science, Carnegie Mellon University, August 1987.
File: **doc/published/threads87{.doc,.ps}**

This paper examines a kernel implemented lightweight process mechanism built for the Mach operating system. The pros and cons of such a mechanism are discussed along with the problems encountered during its implementation.

58. Avadis Tevanian, Jr., Richard F. Rashid, Michael Young, David B. Golub, Mary R. Thompson, William Bolosky, and Richard Sanzi.
A Unix Interface for Shared Memory and Memory Mapped Files Under Mach.
Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, July 1987.
File: **doc/published/mapfiles87{.doc,.ps}**

This paper describes an approach to Unix shared memory and memory mapped files currently in use at CMU under the Mach Operating System. It describes the rationale for Mach's memory sharing and file mapping primitives as well as their impact on other system components and on overall performance.

59. Mary R. Thompson.
Mach Environment Manager.
Unpublished manuscript from School of Computer Science, Carnegie Mellon University. July 1988.
File: **doc/unpublished/envmgr{.doc,.ps}**

The Environment Manager is a Mach server which facilitates the sharing of named variables between tasks. This paper discusses types, primitives, and integration with the Unix environment.

60. Linda R. Walmer and Mary R. Thompson.
A Programmer's Guide to the Mach System Calls.
February 1988. School of Computer Science, Carnegie Mellon University.
File: **doc/unpublished/machsys{.doc,.ps}**

This document is one of two tutorials designed to teach basic Mach programming skills. This manual explains the use of the Mach kernel calls. The companion document to this one, A Programmer's Guide to the Mach User Environment explains the use of higher level methods for implementing multithreaded programs and interprocess communication. Before writing programs that use the system calls directly, the user should be aware that the methods outlined in the other document may be used to solve his problem more simply.

61. Linda R. Walmer and Mary R. Thompson.
A Programmer's Guide to the Mach User Environment.
February 1988. School of Computer Science, Carnegie Mellon University.
File: **doc/unpublished/machuse{.doc,.ps}**

This document is one of two tutorials designed to teach basic Mach programming skills. This manual demonstrates the use of the C Threads library primitives in writing a multithreaded program and the use of the Mach Interface Generator (MIG) to generate remote procedure calls for interprocess communication. The reader should be familiar with the basic Mach abstractions of ports, messages, virtual memory, tasks and threads which the introduction to the companion document, A Programmer's Guide to the Mach System Calls, explains.

62. James W. Wendorf.
Operating System/Application Concurrency in Tightly Coupled Multi-Processor Systems.
Thesis CMU-CS-88-117, School of Computer Science, Carnegie Mellon University, December 1987

At the heart of the thesis are three detailed experimental studies, involving the three major forms of OS/application concurrency. These studies, conducted on a VAX-11/784 shared memory multiprocessor using modified versions of the Mach operating system, explore the practical issues and problems involved in exploiting OS/application concurrency, and demonstrate some feasible approaches and solutions. The first study involves a software specialized IPC processor that can significantly improve the performance communication processes, by exploiting strong concurrency. The second study looks at the scheduling of OS processing on a multiprocessor system, and shoes how performance can be improved through policies that increase the amount of weak concurrency. The final study is an example of global management concurrency, in which the overhead of a complete real-time scheduling policy is significantly reduced by means of a software specialized processor.

The results of the experimental studies support our hypothesis that OS/application concurrency provides significant benefits, independent of specialized hardware.

63. Bob Wheeler, Brian N. Bershad.
Consistency Management for Virtually Indexed Caches.
A version of this paper will appear in the Fifth Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS V), October 1992.
File: **doc/published/asplosVM.ps**

A virtually indexed cache can improve performance by allowing cache lookup and address translation to occur in parallel, thus reducing processor cycle time. Unlike physically indexed caches, virtually indexed caches create consistency problems becasue a physical address may be represented in more than one cache line when it has been accessed through more than one virtual address. *Write-back* virtually indexed caches create additional inconsistencies because memory may become stale with respect to the cache.

In this paper we examine the problem of consistency management for a virtually indexed write-back cache. We assume that the hardware does not support intra-cache consistency. We present a model and software implementation strategy for maintaining consistency with vitrtually indexed caches.

We present measurements from an implementation of this model on the HP 9000 Series 700 in the context of the Mach operating system. Our mearsurements show that a virtually indexed cache can be managed with nearly the same cost as that required to manage a physically indexed one, even when used by a virtual memory system that encourages and exploits sharing.

64. Michael Young, Avadis Tevanian, Jr., Richard Rashid, David Golub Jeffrey Eppinger, Jonathan Chew, William Bolosky, David Black, and Robert Baron.
The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System.
Technical Report CMU-CS-87-155, School of Computer Science, Carnegie Mellon University, August 1987. Also in Proceedings of the 11th Symposium on Operating Systems Principles, November, 1987.
File: **doc/published/dual-mem-com.ps**

An important component of the Mach design is the use of memory objects which can be managed either by the kernel or by user programs through a message interface. This feature allows applications such as transaction management systems to participate in decisions regarding secondary storage management and page replacement. This paper explores the goals, design and implementation of Mach and its external memory management facility. The relationship between memory and communication in Mach is examined as it relates to overall performance, applicability of Mach to new multiprocessor architectures, and

65. Michael Wayne Young.
Exporting a User Interface to Memory Management from a Communication-Oriented Operating System.
Thesis CMU-CS-89-202, November 1989.

This dissertation describes the Mach external memory management interface and implementation in detail. The interface design is based on communication: memory objects are communication channels, on which the operating system sends messages to read or write the contents of pages of virtual memory. The implementation addresses the modularity and security issues that arise from exporting this interface to user-level programs. Several systems applications that have been constructed using the Mach interface are described, including a user-level file system, a transaction-based data management system, and a network shared virtual memory service. An evaluation of the Mach system analyzes the cost of using this interface, and shows how it makes a fast path to the main memory cache available to system applications. Experiments also show that the overall performance of the Mach system does not suffer appreciably under normal load, and that this system still outperforms commercially available operating systems under identical benchmarks. The conclusion is that an external memory management interface is a useful, affordable component that should be provided by future operating systems.

66. Masanobu Yuhara, Brian N. Bershad, Chris Maeda, J. Eliot B. Moss.
Efficient Packet Demultiplexing for Multiple Endpoints and Large Messages
Proceedings of the 1994 Winter USENIX Conference, January 1994.

This paper describes a new packet filter mechanism that efficiently dispatches incoming network packets to one of multiple endpoints, for example address spaces. Earlier packet filter systems iteratively applied each installed filter against every incoming packet, resulting in high processing overhead whenever multiple filters existed. Our new packet filter provides an associative match function that enables similar but not identical filters to be combined together into a single filter. The filter mechanism, which we call the Mach Packet Filter (MPF), has been implemented for the Mach 3.0 operating system and is being used to support endpoint-based protocol processing, whereby each address space implements its own suite of network protocols. With large numbers of registered endpoints, MPF outperforms the earlier BSD Packet Filter (BPF) by over a factor of four. MPF also allows a filter program to dispatch fragmented packets, which was quite difficult with previous filter mechanisms.

Available papers on Real-Time Mach

1. Richard F. Rashid, Hideyuki Tokuda.
   Mach: A System Sofware Kernel
   Proceedings of the Symposium on Computational Technology for Flight Vehicles, November 1990.

   The Mach kernel has been ported to a large collection of uniprocessor and multiprocessor systems and have been used to implement commercial operating system environments such as Unix in a performance competitive way. This paper describes the Mach operating system, the way it was used implement both 4.3BSD Unix and the Macintosh operating system and the implications of this technology for real-time and secure programming environments.

   You may order printed copies from the School of Computer Science, Carnegie Mellon University by sending mail to jfm@cs.cmu.edu or by calling 412/268-7656.

2. Hideyuki Tokuda, Tatsuo Nakajima, Prithvi Rao.
   Real-Time Mach: Towards Predictable Real-Time Systems.
   Proceedings of the USENIX 1990 Mach Workshop, October 1990.
   File: **doc/published/RTM.ps**

   Distributed real-time systems play a very important role in our modern society. They are used in aircraft control, communication systems, military command and control systems, factory automation and robotics. However, satisfying the rigid timing requirements of various real-time activities in distributed real-time often requires ad hoc methods to tune the systems runtime behavior.

   The object of Real-Time Mach is to develop a real-time version of the Mach kernel which provides users with a predictable and reliable distributed real-time computing environment. In this paper, we describe a real-time thread model, real-time synchronization, and the ITDS scheduler in Real-Time Mach. We also discuss the implementation issues, a real-time toolset, and the current status of the system.

3. Hideyuki Tokuda.
   RT-Thread Model for Real-Time Mach.
   Submitted to RTSOSS May, 1991.

   Many traditional multitasking real-time executives often support just a simple multitasking model which is almost equivalent to supporting threads since only one physical address space is provided among tasks. In the executives, it is often that the context switching overhead is very small compared to normal time-sharing operating systems like UNIX. However, it fails to explicitly distinguish between real-time and non real-time threads and lacks to specify any timing attributes for real-time threads. For advanced real-time systems, it is very important to provide a proper protection domain, like a virtual address space, for processes or tasks and to support real-time threads for predictable runtime behaviors.

   The objective of our thread model is to provide predictable and reliable distributed real-time computing environment. In the model, we clearly distinguish between a real-time thread and a non-real-time thread. A real-time thread is defined with timing attributes such as a worst case execution time, deadline or period, and other scheduling policy related information.

   In this position paper, we will introduce a real-time thread (R-T Thread) model which had been developed for Real-Time Mach. We first describe thread creation and termination primitives, synchronization primitives, and integrated time-driven scheduler. We also compare the model with other related thread models.

   You can obtain printed copies from the School of Computer Science, Carnegie Mellon University by sending mail to jfm@cs.cmu.edu or by calling 412/268-7656.

4. Hideyuki Tokuda and Tatsuo Nakajima.
   Evaluation of Real-Time Synchronization in Real-Time Mach.

This paper appears in the proceedings for the USENIX 1991 Mach Workshop, October 1991.
File: **doc/published/RTSync.ps**

Real-Time Mach provides real-time thread and real-time synchronization facilites. A real-time thread can be created for a periodic or aperiodic activity with a timing constraint. Threads can be synchronized among them using a real-time version of the monitor based synchronization mechanism with a suitable locking protocol. In Real- Time Mach, we have implemented several locking policies, such as *kernelized monitor, basic priority, priority inheritance protocol, priority ceiling protocol,* and *restartable critical section,* for real-time applications. It can also avoid a unbonded *priority inversion*problem.

In this paper, we describe the real-time synchronization facilities in Real-Time Mach and its implementation and performance evaluation. Our evaluation results demonstrated that a proper choice of locking policy can avoid unbounded priority inversions and improve the processor schedulability for real-time applications.

Mach documents published in 1990 and after. (Non CMU papers)

1. **Mach/4.3BSD A Conservative Approach to parallelization**
   Joseph Boykin and Alan Langerman Computing Systems, Vol.3, No. 1, Winter 1990
2. **Zone Garbage Collection**
   James Van Sciver and Richard F. Rashid Proceedings of the Usenix Mach Conference, October 1990
3. **Extending the Mach External Pager Interface to Accommodate User-Level Page Replacement Policies**
   Pylan McNamee and Katherine Armstrong Proceedings of the Usenix Mach Conference, October 1990
4. **Mach on a Virtually Addressed Cache Architecture**
   Chia Chao, Milon Mackey, and Bart Sears Proceedings of the Usenix Mach Conference, October 1990
5. **A Highly-Parallelized Mach-based Vnode Filesystem**
   Alan Langerman et al. Proceedings of Winter 1990 Usenix Conference, Jan. 1990
6. **A Persistent Distributed Architecture Supported by the Mach Operating System**
   Francis Vaughan et al. Proceedings of the Usenix Mach Conference, October 1990
7. **A Trusted X Window System Server for Trusted Mach**
   Jeremy Epstein and Marvin Shugerman Proceedings of the Usenix Mach Conference, October 1990
8. **Building a Fault-Tolerant System Based on Mach**
   Rong Chen and Tony P. Ng Proceedings of the Usenix Mach Conference, October 1990
9. **Transparent Recovery of Mach Applications**
   Arthur Goldberg et al. Proceedings of the Usenix Mach Conference, October 1990
10. **Fault-Tolerant Computing Based on Mach**
    Ozalp Babaoglu Proceedings of the Usenix Mach Conference, October 1990

Books with substantial information about Mach

1. **Programming Under Mach**
   Joseph Boykin, David Kirschen, Alan Langerman,Susan LoVerso
   Addison-Wesley Publishing Co,. 1993 (481 pages)

This is certainly the most complete presentation of how to program applications using Mach features. It explains the basic Mach abstractions of tasks, threads, ports, messages and memory objects and gives examples of why and how to use them. There are introductory chapters for tasks/threads, interprocess communication (IPC) and virutal memory. Following this are chapters on advanced features of IPC such as server processes might need, the Mach Interface Generator which is used to simplify the writing of IPC interfaces, and external memory management. There is also a chapter each for C-threads and OSF/1 P-threads, and the Mach exception handling facility.

There are frequent short code examples as well as a complete client/server example of an external memory manager.

The code is all compatible with Mach 2.5, but references are made to where there are differences with Mach 3.0. Even if your goal is to program for a Mach 3.0 system, this book is a good place to start since the basic concepts have not changed significantly. Once the basic ideas are understood, the Mach 3.0 reference manuals can be used to update your knowledge of the details.

2. **Distributed Systems, Concepts and Design, 2nd Ed.**
   G. Coulouris, J. Dollimore, T. Kindberg
   Addison-Wesley Publishing Co., 1994

   This is a textbook intended to provide knowledge of the principles and practice of distributed system design. The principles of interprocess communication, remote procedure calls, distributed file and name servers, and distributed operating systems with the emphasis on micro-kernels are covered along with other more sophistcated features to support distributed systems. Many of these ideas are illustrated by examples from the Mach operating system.

   There is a final chapter where Mach, Chorus, Amoba and Clouds are examined in more detail and the different design decisions are compared. Section 18.2, pages 547-566 gives a clear explanation of the central features of the Mach 3.0 micro-kernel.

3. **Distributed Operating Systems: The Logical Design**
   A. Goscinski
   Addison-Wesley Publishers., 1991

   Chapter 14.8, pages 864-888 is about Mach

   This is a subsection of a chapter surveying seven experimental distributed operating systems. It defines the basic Mach abstractions and goals. It emphasizes the software architecture both at the high level of the system tasks running on tightly coupled multi-processor platforms or a network of loosely coupled machines and at the implementation level of describing the various kernel modules and data structures. Specific primitives are specified and an outline of a client/server shared memory server is given.

4. **Load Distribution, Implementation for the Mach Microkernel**
   Dejan S. Milojicic, OSF Research Institute, Cambridge, Mass.
   Verlag Publishing, Wiesbaden, Germany 1994

   The book is of practical character and describes the author's experience while developing load distribution on top of the Mach microkernel.

   Load distribution is a very important concept for distributed systems in order to achieve better performance, resource utilization and response times. Providing efficient mechanisms for the transparent support of load distribution has proven to be an extremely difficult undertaking. As a matter of fact, there is no commercially available system which provides transparent load distribution right now. The monograph by D. Milojicic presents a novel load distribution scheme based on modern microkernel architectures. The remarkable results of D. Milojicic's approach show evidence for his hypothesis that load distribution is feasible even under strong efficiency constraints if built upon microkernel architectures. Based on a complete implementation using the NORMA-version of Mach, D. Milojicic shows that substantial performance improvements of his load distribution scheme on top of Mach result from the dramatic reduction of state information to be managed in course of a task migration.

For readers not familiar with the topic, the monograph gives a good survey of the load distribution problem and puts existing approaches into perspective.

Abstact is from the forward by Prof. Juergen Nehmer, University of Kaiserslautern

5. **CMU Computer Science: A 25th Anniversary Commemorative**
   Richard F. Rashid
   Mach: A Case Study in Technology Transfer
   Chapter 17, pgs. 411-421
   ACM Press, Anthology Series, 1991

   This chapter describes history of the Mach operating system research project and its effort to transfer the ideas imbedded in that system from the research laboratory to industry.

6. **The Mach Operating System.**
   Abraham Silberschatz, James L. Peterson, Peter B. Galvin
   Operating Systems Concepts, Third Edition
   Chapter 16, pgs. 597-628
   Addison-Wesley Publishing Company

   Gives a brief introduction to Mach, its history and design goals. Covers the basic concepts of Mach, its similarities and differences from Unix. Discusses the NetMessage Server and how it extends IPC trasparently across machines. Covers virtual memory managment and external memory managers and sharing of memory between tasks. There is a brief mention of implementation details and how they compare with BSD-Unix.

7. **Modern Operating Systems**
   Andrew S. Tanenbaum
   Prentice Hall 1992
   Chapter 15, pages 637-682 (Case Study 4) is about Mach

   This chapter gives a general introduction to the Mach 3.0 micro-kernel and compares it to the Amobea operating system. Implementation of task and thread creation and scheduling are covered in some depth as is memory management and interprocess communication. External memory management is explained and there is a brief description of the Network Message Server and the Unix server.

8. **Mach: Konzepte und Programmierung**
   Christoph Zimmermann, Albrecht W. Krass
   German edition, Springer-Verlag Publishing Company, 1993 (191 pages)

   This book covers the concepts and programming interfaces of the Mach kernel. It addresses both the novice and the more advanced reader. The first part reviews the reader’s knowledge of operating systems, and introduces the basic Mach abstractions of VM objects, threads, tasks, etc...

   The second part presents an in-depth description of the following topics: Mach IPC, the MiG, Cthreads, the Mach VM system, the External Memory Management Interface (EMMI). An application example gives this section on Mach programming final polish. An outlook in the last chapter presents the differences between the 2.5 version of Mach and the Micro-Kernel based 3.0 version, together with the description of the Unix-Emulation.

The following documents were edited by Keith Loepere of the Open Software Foundation, and have been distributed at various OSF/RI Mach workshops. Thus you may already have copies of them. They all document Mach 3.0, but in the case of the Kernel Principles manual and the Server Writer’s Guide much of the information is applicable to Mach 2.5. If you are interested in obtaining hard copies of the documentation please contact Susan Crocker vi\ a e-mail <crocker@osf.org> or by phone at 617/621-7259.

1. Keith Loepere.
   Mach 3 Kernel Principles.
   Open Software Foundation document.
   File: **doc/osf/kernel_principles.ps**

   Documents the user visible architecture of the Mach 3.0 kernel. It is currently the best detailed introductio\ n to the basic Mach concepts: tasks, threads, ports, messages, memory objects, virtual address spaces, processors, hosts and devices. (52 pages)

2. Keith Loepere.
   Mach 3 Kernel Interface.
   Open Software Foundation document.
   File: **doc/osf/kernel_interface.{1,2,3,}.ps**

   Describes each of the Mach 3.0 kernel primitives in isolation. The format is that of a man page per kernel call. All introductory material is in "Mach 3 Kernel Principles" Total document is 355 pages. Each ps file is about 100+ pages.

3. Keith Loepere.
   Server Writer’s Guide.
   Open Software Foundation document.
   File: **doc/osf/server_writer.ps**

   Documents the major user level tools for writers of Mach servers: MIG and CThreads. Contains descriptive material and examples. Similar to the Mach 2.5 documents, "MIG - The MACH Interface Generator", Draves, et.al. and "C Threads", Cooper, Draves, but updated to reflect the changes for Mach 3.0 (44 pages)

4. Keith Loepere.
   Server Writer’s Interface.
   Open Software Foundation document.
   File: **doc/osf/server_interface.ps**

   Describes each of the CThreads primitives in isolation. Introductory material is all in the "Server Writer’s Guide". (52 pages)