The Increasing Irrelevance of IPC Performance for Microkernel-Based Operating Systems

Brian N. Bershad
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Brian.Bershad@cs.cmu.edu

March 10, 1992

Abstract

IPC is the glue with which 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 IPC performance is becoming increasingly irrelevant to overall system performance.

1 Introduction

Microkernels such as V [Cheriton 84], Chorus [Rozier et al. 88], and Mach [Accetta et al. 86] provide the infrastructure with which other operating systems such as Unix, MS-DOS and VMS can be implemented as user-level programs [Golub et al. 90, Rashid et al. 91, Cheriton et al. 90, Wiecek 92]. Because applications use cross-address space IPC to interact with traditional operating system services, IPC performance has been thought to be the Achilles heel of microkernel-based systems. Even with the improvements in IPC performance that have occurred in the past 5 years [Bershad et al. 90, Draves et al. 91], microkernel based systems are still believed to have *inherently* worse performance than their monolithic counterparts. This belief stems from the fact that the system call, which is the mechanism for interacting with monolithic operating systems, such as

This research was sponsored in part by The Defense Advanced Research Projects Agency, Information Science and Technology Office, under the title "Research on Parallel Computing", ARPA Order No. 7330, issued by DARPA/CMO under Contract MDA972-90-C-0035 and in part by the Open Software Foundation (OSF), and a National Science Foundation Presidential Young Investigator Award.

4.3BSD [Leffler et al. 89] and Sprite [Ousterhout et al. 88], is faster than a cross-address space IPC, which is how applications interact with user-level operating system services.

Although IPC latency in microkernels is slower than system call latency, the absolute difference between them has reached the point where it can be largely ignored. In other words, IPC performance is becoming increasingly irrelevant as a metric with which to assess microkernel viability.

There are four reasons that IPC performance should no longer be a principal metric by which one judges the "goodness" of a particular operating system microkernel, or even a particular approach to building operating system microkernels. In brief, these reasons are:

1. IPC has gotten faster faster than the rest of the operating system.
2. Performance is dominated by caches, not by address spaces.
3. All data does not need to be marshalled through the kernel.
4. All services do not need a hardware firewall.

The first two reasons stem from the ever-growing performance imbalances which exist in today's systems. Simply stated, IPC mechanisms are not the performance bottlenecks which they once were. Instead, there are other, more fundamental bottlenecks, such as memory transfer speed, network latencies, disk speed, and cache management overhead. The second two reasons are due to the maturity which microkernel-based systems have achieved in the last few years. Specifically, in their efforts to increase performance, systems builders have discovered a collection of techniques which can be used to bypass, or at least "dance around" IPC facilities.

In the rest of this paper I expand on the reasons listed above, and discuss why we should stop measuring microkernel systems by the speed of their round trip IPC times. I present examples and observations from the Mach 3.0 microkernel running on a collection of architectural platforms to motivate and substantiate the discussion.

2 IPC Has Gotten Faster Faster Than the Rest of The Operating System

It has been observed that operating system performance has improved far less rapidly than would be expected given improvements in processor architecture and implementation [Ousterhout 90, Anderson et al. 91]. Although the time to add two registers together has decreased by almost two orders of magnitude in the last decade, the performance of key operating system services, such as filing, paging, and networking, has remained relatively flat. Disks continue to spin at about the same speed as they always have, buffer caches remain limited by memory bandwidth, and core network latency remains on the order of several hundred microseconds (although network bandwidth has improved somewhat more dramatically).

In contrast to the performance of the services which are being provided by the operating system, the time to send a message between two address spaces on the same machine has dropped substantially. The reasons for this are two-fold. First, we've become more careful and more successful at building IPC mechanisms "for speed," ruthlessly streamlining and optimizing the common cases. Using the Mach 3.0 microkernel as an example, IPC performance on a Microvax-III (CVax processor) has gone from about 750 μsecs for a round-trip RPC in 1989 [Bershad 90] to 497 μsecs in 1992 (measured recently using Mach 3.0 version MK68). The improvements were due to tightening the interface [Draves 90], and the implementation [Draves et al. 91].

The second reason why IPC has gotten faster faster than the rest of the operating system is that measured IPC performance has, at least to this point, tracked processor performance reasonably well.[1] In Mach, this is largely because the improvements made in the last several years have moved IPC performance off of the memory curve and onto the processor curve by tightening the locality of the IPC paths. Again, using Mach 3.0 MK68, a round-trip RPC takes 57 μsecs[2] on a DecStation 5000/200. That system, which uses a MIPS R3000 processor running at 25Mhz, is rated at roughly ten times the performance of the CVax. In keeping with this, cross-address space RPC is about nine times faster than the same code running on a CVax.

With IPC performance improving more rapidly than general operating system service performance, the "hit" required to access such a service is becoming less important. For example, on the older CVax, the 497 μsecs required to do a cross-address space RPC was comparable to the time it took seek one disk track, or copy a 512 byte block from a system buffer cache into a user buffer, or to transmit a packet over an Ethernet. On the DecStation 5000/200, however, the IPC penalty is substantially less. Since disk access, data transfer, and network latencies have not improved at anywhere the same rate, the additional cost of the IPC required to indirect to these facilities has become less significant.

2.1 System Calls Are Not The Solution

System calls used in monolithic kernels are the alternative to the IPC mechanisms used in microkernels. Monolithic operating system services reside in the kernel, and are accessed by applications with a single kernel boundary crossing. On older systems, such as the CVax, the time to execute a Mach system call[3] was about 60 μsecs, substantially less than the IPC overhead (497 μsecs) on that machine. In contrast, on the newer DecStation 5000/200, system call overhead is about 8 μsecs, or about 50 μsecs less than a round-trip RPC. While the relative cost of IPC and system calls on the two machines is the same, the absolute difference *compared* to the service access cost (disk, buffer, or network), is much smaller. As a result, there is diminishing incentive to use system calls, rather than generalized IPC facilities, to access system services.

3 Performance is Dominated By Caches, Not by Address Spaces

The invocation of an operating system service implies a change in locality. In monolithic systems, the new locality is in the kernel. In microkernel-based systems, the new locality is in another address space. In both cases, however, the change in locality can result in an increased cache miss rate [Agarwal et al. 88, Mogul & Borg 91]. On older style, slower architectures (< 10 MIPS), cache miss penalties were only a few cycles. On today's architectures, however, cache miss penalties are tens, and soon to be hundreds, of cycles. These kinds of penalties can easily dwarf the kernel's IPC overhead. In effect, the cost of accessing an operating system service is going to be most influenced by whether or not that service is in the cache – not whether or not it's in the kernel or in another address space.

One could argue, however, that OS interaction via a microkernel actually involves two changes in locality – one to the microkernel and another to the server. While true,

[1] Although IPC performance is ultimately limited by architectural features such as trap and context switch time, and although these times have not been improving at the same rate as processor speed [Anderson et al. 91], practical IPC performance has not yet reached this limit.

[2] All times were measured with warm caches.

[3] Mach, although a microkernel, does export a small number of "true" system calls.

the microkernel's locality (at least on the critical path through to the operating system server) is small. The common-case round trip IPC path in Mach 3.0 on the MIPS, for example, requires less than 4KB of instructions and references less than 2KB of data, most of which is on the kernel stack. Because the locality is small, and identifiable *a priori*, one could decrease the chance that misses occur on the IPC path by allocating memory with an eye towards the cache hashing function [Bershad et al. 92]. For virtually addressed caches, this means devoting pieces of the machine's virtual address space to the microkernel. For a physically addressed cache, physical memory must be devoted, although one can play tricks with split instruction and data caches to ensure that only data pages conflict with designated instruction pages, and vice versa.

One place where cache performance becomes apparent is in the management of external devices which use DMA, and not programmed, I/O. Before the processor reads memory into which DMA has been performed, it must purge that memory from the cache to ensure that stale data is not returned. Before a processor issues a write request to a DMA device, it must flush the memory to be written from the cache, again to ensure that stale data is not read by the device. These cache operations are expensive. For example, on the HP-700, a high performance workstation based on the HPPA RISC processor with a cycle time of 20 nanoseconds, cache purge and flush operations take between 1 and 14 cycles per line (32 bytes), DMA operations tend to be page-oriented, so I/O operations require between 128 and 896 cycles simply to ensure memory coherency. In the case of device reads, performance degrades even further as the newly transferred data is faulted into the cache, at a cost of 16 cycles per line (2048 cycles per page). In contrast, a cross-address space RPC in Mach 3.0 on that machine takes about 70 μsecs (3600 cycles). Consequently, the *cost* of accessing an out-of-kernel device server (changing address spaces) represents only one component of the total CPU/device communication cost.

4 All Data Does Not Need to be Marshalled Through the Kernel

Microkernel-based operating systems can preallocate buffers between client and server address spaces. This allows an operating system service to share address space with applications, just as it did when the service was resident in the kernel. Small to medium amounts of data can be transferred from one address space to another by depositing it in the shared regions, rather than by sending it through the kernel in a message [Bershad et al. 91]. Large data segments (on the order of pages) can be passed using virtual memory primitives.

Co-mapping of IPC data has been used in several places. At CMU, applications share memory with the Unix server to pass data in and out of the file system. We have recently applied this technique to the socket interface as well. On a DS5000/200, we have found that the mapped socket interface does not improve the performance on small packets (fewer than 100 bytes), but in larger packets there is an improvement. For packets of 4KB, sends using the mapped socket interface are 15% faster than using the regular, non-mapped, interface. In packets of more than 4KB the mapped interface avoids the cost of dynamically allocating and deallocating the region for out-of-line transmission, although for extremely large packets remapping, rather than copying, is more efficient.

Co-mapping can be used effectively for non-Unix interfaces as well. For example, we have built a version of the X11 window server which uses Mach IPC for communication between X clients and the server. X does call request buffering, but clients frequently flush the buffer, which causes the data to be made available to the server. When Unix stream sockets are used to implement the transport layer, the flush causes a socket write to occur. With sockets, multiple flushes may occur before the server collects any of the data, but all of the data can be collected in one receive, so transferring data through the kernel doesn't necessarily increase the number of context switches. When using Mach's IPC, which is message-oriented, every flush results in a message. Message boundaries are maintained, so every message must be collected separately by the X server, increasing the the number of user-kernel boundary crossings and context switches relative to a socket-based implementation. We are modifying the X library to buffer requests in shared memory. Flushes transfer the data to the shared buffer, and only cause an IPC message to be sent if previous IPC messages have not yet been collected.

5 All Services Do Not Need a Hardware Firewall

The cost of accessing functions in another protection domain (whether in the kernel or in a server's address space) has provided motivation for microkernel-based systems to migrate what were once kernel functions into client address spaces. This is possible when unprotecting the functions has no security implications. For example, there's no reason to put the system clock in the kernel, let alone another address space – on most architectures, the clock can be mapped directly into user address spaces. Similarly, network protocols can execute in the address spaces of the applications which are communicating rather than in the kernel or a special protocol server [Schroeder & Burrows 90, Maeda & Bershad 92]. Applications can send and receive packets directly through the network interface. If the network is assumed to be insecure (as is generally the case), then executing the protocols in a secure protection domain offers no additional integrity. Encryption, and not hardware protection modes, are necessary here.

Another technique for removing hardware firewalls is to migrate pieces of the operating system service into clients' address spaces. Mach's Unix emulation package uses this approach with its transparent emulation library, which is a shared library mapped into every Unix address space. Unix system calls are reflected out of the kernel into the caller's emulation library. There, the emulation library may simply forward the system call onto the Unix server, or it may implement the call itself, if possible. For example, the emulation library uses a mapped file interface for communicating with the Unix server. Read and write system calls are intercepted by the emulation library and converted into loads and stores to the mapped memory which backs the file. In this way, binary compatibility with Unix is maintained, and cross-address space IPC is avoided. This approach has been generalized in the Mach multiserver project [Julin et al. 91], and has resulted in substantial IPC reductions. In benchmarks on that system, which are intended to be persuasive but not conclusive, client-side emulation permits two out of three system calls to be implemented without an RPC.

6 Conclusions

IPC performance has come a long way in the last ten years. We now understand how to build IPC mechanisms which are only a few tens of microseconds slower than system calls. While this may at first seem unacceptably high, an examination of the other issues in operating system performance reveals that the additional overhead is small compared to the services which are being accessed. Moreover, the growing mismatch between cache and memory speed is making the physical location of operating system code and data much more important than the software path by which it is accessed. Finally, as microkernel-based operating systems have matured, useful techniques which reduce the frequency of operating system interaction, and hence, IPC, have also been developed. For these reasons, the raw performance of IPC facilities is largely becoming an irrelevant metric by which to judge microkernel-based operating systems.

Acknowledgements

Jose Carlos Brustoloni, Rich Draves, Dan Julin, Mark Stevenson, and Bob Wheeler supplied most of the measurements presented in this paper. Conversations with these people, as well as with Bob Baron, Joe Barrera, Alessandro Forin, Michael Ginsberg, Chris Maeda and Dan Stodolsky contributed to this paper.

References

[Accetta et al. 86] Accetta, M. J., Baron, R. V., Bolosky, W., Golub, D. B., Rashid, R. F., Tevanian, Jr., A., and Young, M. W. Mach: A New Kernel Foundation for UNIX Development. In *Proceedings of the Summer 1986 USENIX Conference*, pages 93–113, July 1986.

[Agarwal et al. 88] Agarwal, A., Hennessy, J., and Horowitz, M. Cache Performacne of Operating System and Multiprogramming Workloads. *ACM Transactions on Computer Systems*, 6(4):393–431, November 1988.

[Anderson et al. 91] Anderson, T., Levy, H., Bershad, B., and Lazowska, E. The Interaction of Architecture and Operating System Design. In *Proceedings of the Fourth Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 108–121, April 1991.

[Bershad 90] Bershad, B. N. *High Performance Cross-Address Space Communication.* PhD dissertation, University of Washington, Department of Computer Science and Engineering, Seattle, WA 98195, June 1990.

[Bershad et al. 90] Bershad, B. N., Anderson, T. E., Lazowska, E. D., and Levy, H. M. Lightweight Remote Procedure Call. *ACM Transactions on Computer Systems*, 8(1):37–55, February 1990. Also appeared in *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, December 1989.

[Bershad et al. 91] Bershad, B. N., Anderson, T. E., Lazowska, E. D., and Levy, H. M. User-Level Remote Procedure Call. *ACM Transactions on Computer Systems*, 9(2):175–198, May 1991.

[Bershad et al. 92] Bershad, B. N., Forin, A., and Draves, R. Cache Effects for a Microkernel Operating System. Technical report, School of Computer Science, Carnegie Mellon University, 1992. In preparation.

[Cheriton 84] Cheriton, D. R. The V Kernel: A Software Base for Distributed Systems. *IEEE Software*, 1(2):19–42, April 1984.

[Cheriton et al. 90] Cheriton, D. R., Whitehead, G. R., and Sznyter, E. W. Binary Emulation of Unix using the V Kernel. In *Summer 1990 Usenix Conference Proceedings*, 1990.

[Draves 90] Draves, R. P. A Revised IPC Interface. In *Proceedings of the First Mach USENIX Workshop*, pages 101–121, October 1990.

[Draves et al. 91] Draves, R. P., Bershad, B. N., Rashid, R. F., and Dean, R. W. Using Continuations to Implement Thread Management and Communication in Operating Systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 122–136, October 1991.

[Golub et al. 90] Golub, D., Dean, R., Forin, A., and Rashid, R. Unix as an Application Program. In *Proceedings of the Summer 1990 USENIX Conference*, pages 87–95, June 1990.

[Julin et al. 91] Julin, D. P., Chew, J. J., Stevenson, J. M., Guedes, P., Neves, P., and Roy, P. Generalized Emulation Services for Mach 3.0: Overview, Experiences and Current Status. In *Proceedings of the 1991 Usenix Mach Workshop*, November 1991.

[Leffler et al. 89] Leffler, S., McKusick, M., Karels, M., and Quarterman, J. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, Reading, MA, 1989.

[Maeda & Bershad 92] Maeda, C. and Bershad, B. N. Networking Performance for Microkernels. In *Proceedings of the Third Workshop on Workstation Operating Systems*, April 1992.

[Mogul & Borg 91] Mogul, J. and Borg, A. The Effect of Context Switches on Cache Performance. In *Proceedings of the Fourth Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 75–84, April 1991.

[Ousterhout 90] Ousterhout, J. K. Why Operating Systems Aren't Getting Faster As Fast As Hardware. In *Proceedings of the summer 1991 USENIX Conference*, pages 247–256, June 1990.

[Ousterhout et al. 88] Ousterhout, J., Cherenson, A., and Douglis, F. The Sprite Network Operating System. *IEEE Computer Magazine*, 21(2):23–36, February 1988.

[Rashid et al. 91] Rashid, R. F., Malan, G., Golub, D., and Baron, R. DOS as a Mach 3.0 Application. In *Proceedings of the 1991 Usenix Mach Workshop*, pages 27–40, November 1991.

[Rozier et al. 88] Rozier, M., Abrossimov, V., Armand, F., Boule, I., Giend, M., Guillemont, M., Herrmann, F., Leonard, P., Langlois, S., and Neuhauser, W. The Chorus Distributed Operating System. *Computing Systems*, 1(4), 1988.

[Schroeder & Burrows 90] Schroeder, M. D. and Burrows, M. Performance of Firefly RPC. *ACM Transactions on Computer Systems*, 8(1):1–17, February 1990.

[Wiecek 92] Wiecek, C. A Model and Prototype of VMS Using the Mach 3.0 Kernel. In *Proceedings of the 1992 Usenix Microkernel Workshop*, April 27–28 1992. This issue.