Colocating Servers

Burt Rosenberg
Department of Math and Computer Science
Univ. of Miami, Florida
burt@cs.miami.edu

1 Introduction

Just as Mach derived microkernels move into the marketplace, several sources have been indicating why this event has been so slow in coming. Mach achieved protection through capability based message passing. In practice, this placed a small operating system's tax on communication. One supposed that as cycles became cheaper, the benefits would outweight the cost. But this has not been the case: the inefficiency of message passing has remained an issue, message passing directly competes for cycles with new software units that one is pressed to implement, and faster computers may even exacerbate the situation.

Making microkernels a market reality required clever implementation and insightful rethink. The two costs incurred in message passing are the context switch and the data copy. The context switch is particularly heavy, since it crosses the kernel-user protection boundary. Solutions include the OSF Research Institute's introduced colocation and a thread migration model [11], user space message passing [5][6], shared-memory message passing [10] and the use of continuations to reduced the cost of context switch during message hand-off [12].

Some of these approaches succeed by repeating kernel functionality external to the kernel, incurring a one time cost to move a resource to the side of a protection boundary where it will be repeatedly used. This idea is also found in user-level threading packages [22]. Other approaches succeed by moving objects into the kernel in a more or less automatic fashion, thus preserving both microkernel advantages and integrated kernel efficiency. The OSF server colocation [11], the Cache Kernel [9], and the SPIN project [4] are examples of this approach. Finally, it has been proposed to demolish entirely these boundaries as kernel entities, as in the Exokernel [13] [14][15].

We propose an operating system with object colocation as a basic principle and plan to combine it with an Exokernel-like minimalist kernel. We shall describe object colocation in relationship to our target application: a multi-server operating system architecture. Examples of these architectures are found in Chorus [7], Mach-US [23], and Spring [17].

In our architecture, each service is an object which is colocated into the client's world. Opening a server establishes a connection to the object implementing that service and a thread migrates from the server into the client's space. The caller will have a dynamic client-stub binder, called the Object Broker, to handle the migration. The migrated thread will implant in the user's address space, running both in the user's space and in a protected area which allows for communcication back to the server, and between server threads, using shared-memory and address-valued signals [10]. When invoked, the migrated thread will use the invoker's resources and context, thus reducing the price of service invocation to almost that of a procedure call. The additional work includes widening the process's memory privileges. We propose doing this fast by leveraging the memory management index registers.

The paper is structured as follows. Section two presents a mathematical analysis of message passing costs. Section three discusses server colocation. The last section gives our conclusions and future directions.

2 Analysis of Communication Costs

We present an argument for the continued importance of IPC costs in distributed computing. Our analysis makes certain assumptions which yield a model parameterized by processing power. Let p be processing power, measured, say, in the number of CPU cycles per second. The processing power is divided among n concurrently running processes. It is necessary, but not straight-forward, to correctly estimate n. The writer of a "killer app" would like to set n to one. For the sake of computing n, a single process is distinguished by little uncertainty as to the outcome of the computation. Non-frivolous communication with the outside will cause branching in the computation which could not be predicted from inside the process. Hence the communication was required.

As an extreme case, we posit that processing power devoted to communication grows as αn^2, for some small constant α. Among n processes, there are about $n^2/2$ possible process interactions. We suppose that some fixed fraction of all possible interactions result in an interaction. The constant α summarizes both this preference for interaction and the cost of the communication. The processing power devoted to non-communication tasks is a constant β, since it essentially computes a predetermined output given the information gained by communication. We have the following equation of balance for processing power,

$$p = \beta n + \alpha n^2.$$

Increasing n, the second term dominates,

$$p = \alpha n.$$

Hence, for fixed, large p the operating system "tax" on communication α and "other uses" for computation cycles are in direct competition. The tax does not become negligible, it always represents foregone opportunities to use cycles elsewhere.

Our assumptions are extreme. We rewrite our equation of balance more generally,

$$p = \beta(n)n + \alpha(n)n,$$

where α and β are now functions of n. Our previous paragraph set β constant and α proportional to n, yet all that is required is that,

$$\beta(n) = o(\alpha(n)).$$

For instance, perhaps as n grows, the amount of straight-line computing inside a process grows as $\log n$, as more information can be squeezed out of the communication interactions, and perhaps a linear increase in opportunities to communicate results in only $\sqrt{n}$ more interactions. The conclusion still holds.

The parameterization of the model by computing power distinguishes our analysis from Bershad's [3]. His argument applies after the current wave of adjustments to α and β, treated as constants for a fixed n. Growth of p will require additional fixing of the constants, which might or might not be feasible.

3 Colocating Services

Running what were traditionally kernel services in user space introduces additional costs. For instance, crossing the kernel boundary is expensive, and both control and data must cross this boundary. A form of cacheing can be employed, where one time start up costs are paid to minimize recurring usage fees. Each service has an implementing server containing a pool of a perhaps unlimited number of threads. To gain access to a service, a server thread is migrated into the client's space. Each future request to the server is performed using this thread, and server usage cost can almost be reduced to that of a procedure call.

The pioneering implementation of this form occurs in OSF's UNIX port [11]. Here servers colocate into the kernel. Those researchers did note that colocation among user processes is also a possibility. We follow up on this possibility, creating a multi-server operating system running on top of an extremely lightweight kernel such as the Exokernel [13] [14] [15].

3.1 Shared Memory

The colocated services use shared memory which is protected from the hosting process to communicate with the hosting process and among the server threads. The use of shared memory and address-valued signal (see Cheriton [10]) is a crucial component in this architecture. It is their value, when fully exploited, that this architecture will determine.

The virtual address space of the hosting process is divided into three regions. The lower half belongs to the process. The upper half is divided into a protected region which is owned by the currently active colocated server thread, and kernel memory. Kernel memory is not accessible to either the process or the server thread. This structure achieves a balance between protection and data access which generalizes the kernel-user space hierarchy of the VAX implementation of 4.3BSD virtual memory [18].

The protected region is constantly remapped, as various colocated server threads become active. Hence during the lifetime of the process, its virtual memory contents changes constantly. Our viewpoint of migrating threads includes a more fluid conception of virtual memory than one typically has in either traditional Unix or Mach. When migrating a thread into the hosting process, the setup for the virtual memory follows a separate route than does the thread: it is sent directly to the memory management software, with a tag for later identification.

3.2 The Object Broker

The object broker dynamic binds client stubs into the user's address space. It is linked into the user program and is part of every application's runtime support. It is responsible for choosing and locating services, opening a connection to the server, receiving the server thread, choosing a mapping of the thread's address space into the user's virtual address space and linking the thread there, and invoking the thread with each service call. To the application, the object broker is a catalog of service threads, and forms the operating system personality as is apparent to the application.

The text of the server thread may be position independent code. In which case, the object broker has the possibility of simply linking the code into the lower half of the hosting process's virtual address space. Else, the thread will be written to access the protected region for both private data and text. The process identification number for the originating server forms the high order bits of this memory, viewing the entire memory space of the operating system as a single, very large virtual address space.

Upon a call to the service thread, the few high order bits of the process's virtual address space which map to the protected region are set to refer to a the region of the global virtual address space in which the service thread is positioned. The virtual memory system has been informed of this region by the originating server, and the proper backing store for this region, be it a vnode for text, or anonymous memory for data, has been installed. The virtual memory support required is standard in modern systems such as Solaris and Mach.

3.3 Assumptions about Hardware

We claim that the use of hardware resources such as the TLB's in the MIPS 4000 and memory index registers in the PowerPC will provide an efficient way of remapping the virtual memory spaces as each server thread runs and returns. We refer to this remapping as privilege widening. The return from server code requires an inverse privilege narrowing.

For example, the top four address bits of the 32 bit Logical Address of the PowerPC 601 are mapped by 16 on chip Segment Registers to 24 bit Virtual Segment ID's [21]. These ID's are used to locate the Physical Page number in the in core page table. Simply filling the four segment registers 12 through 15 will remap the privilege as required.

This implies that our approach is not architecture neutral: it favors large virtual address or segmented and tagged address space architectures, where CPU loadable memory indirection registers select for a memory subspace or tag. Our hardware model includes these *memory mapping registers,* of which the MIPS and PowerPC architectures are conforming examples. It is possible that future kernels will include an abstraction in support of this memory model.

4 Conclusions and Future Work

We have proposed an operating system architecture based on the colocation of server threads. We feel that this leads to efficient operating systems using microkernel and multi-server technologies. Our insistence on reducing communication costs was backed by a mathematical model. This model proposed a condition on the functional form of communication and computation growth versus system complexity and processing power. It should be possible to observe these functions from historical data, and we look forward to this as future work.

The solution proposed made heavy use of shared memory and proposed hardware assist in the use of TLB's to make the solution viable. The use of TLB's as a structuring principle in operating systems architectures is an avenue for future research.

Finally, it would be extremely instructive to implement an operating system based on colocation on top of an extremely light-weight kernel, such as the Exokernel. We are considering the requirements to realize this project, and are looking forward to moving ahead with our implementation.

Acknowledgements

The author thanks IBM for equipment and software grants for this project, in particular, the support of Izzy Magrisso at IBM, Bota Raton and the IBM Microkernel Group; also Chip Elliott for important contributions to the ideas and presentation of ideas in this paper.

References

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, M. Young, *Mach: a new kernel foundation for Unix development,* Proceedings Summer 1986 USENIX Conference, July 1986. Pp. 93–112.

[2] F. Armand, F. Herrmann, J. Lipkis, M. Rozier, *Multi-threaded processes in CHORUS/MiX,* Proc. of EUUG Spring'90 Conference, Munich, Germany, 23-27 April 1990, pp.1-13

[3] B. N. Bershad, *The increasing irrelevance of IPC performance for Microkernel based operating systems,* Proceedings of the USENIX Workshop on microkernels and other kernel architectures, USENIX, April 1992. Pp. 205–212.

[4] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, C. Chambers, S. Eggers, *Extensibility, safety and performance in the SPIN operating system,* Proceedings of the Fifteenth Symposium on Operating Systems Principles, Dec. 1995, Pp. 267–284.

[5] B. N. Bershad, T. E. Anderson, E. D. Lazowska, H. M. Levy, *Lightweight remote procedure call,* ACM Transactions on Computer Systems, Vol. 8, No. 1, February 1990. Pp. 37–55.

[6] B. N. Bershad, T. E. Anderson, E. D. Lazowska, H. M. Levy, *User-level interprocess communication for shared memory multiprocessors,* ACM Transactions on Computer Systems, Vol. 9, No. 2, May 1991. Pp. 175–198.

[7] A. Bricker, M. Gien, M. Guillemont, J. Lipkis, D. Orr and M. Rozier, *A new look at micro-kernel-based UNIX operating systems: Lessons in performance and compatibility,* Proc. of the EurOpen Spring'91 Conference, Tromsoe, Norway, 20-24 May 1991.

[8] D. R. Cheriton, *The V distributed system,* Communications of the ACM, 1988. Pp 314–333.

[9] D. R. Cheriton, K. J. Duda, *A caching model of operating system kernel functionality,* First Symposium on Operating Systems Design and Implementation, Nov. 1994, Pp 179–193.

[10] D. R. Cheriton, R. A. Kutter, *Optimized memory-based messaging: Leveraging the memory system for high-performance communication,* Stanford University Computer Science Technical Report, CS–TR–94–1506, 1994.

[11] Michael Condict, Dave Mitchell, Frankline Reynolds, *Optimizing performance of Mach-based systems by server co-location: A detailed design,* OSF Research Institute, Auguest 1993.

[12] R. P. Draves, B. N. Bershad, R. F. Rashid, R. W. Dean, *Using continuations to implement thread management and communication in operating systems,* Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles, 1991.

[13] D. R. Engler, M. F. Kaashoek, J. O'Toole, Jr., *The operating system kernel as a secure programmable machine,* Proceedings of the Sixth SOGOPS European Workshop, Sept. 1994.

[14] D. R. Engler, M. F. Kaashoek, J. O'Toole, Jr., *The Exokernel approach to operating system extensibility,* Proceedings of the First Symposium on Operating Systems Design and Implementation, Nov. 1994, Page 198.

[15] D. R. Engler, M. F. Kaashoek, J. O'Toole, Jr., *Exokernel: An operating system architecture for application-level resource management,* Proceedings of the Fifteenth Symposium on Operating Systems Principles, Dec. 1995, Pp. 251–266.

[16] G. Hamilton, P. Kougiouris, *The Spring nucleus: A microkernel for objects,* Sun Microsystems Laboratories, SMLI TR–93–14, April 1993.

[17] Y. A. Khalidi, M. N. Nelson, *An implementation of Unix on an object-oriented operating system,* Sun Microsystems Laboratories, SMLI TR–92–3, December 1992.

[18] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels John S. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System,* Addison-Wesley, 1989.

[19] K. Loepere, *OSF Mach Final Draft Kernel Principles,* Open Software Foundation and Carnegie Mellon University, 1993.

[20] C. Maeda, B. N. Bershad, *Service without Servers*, CMU–CS–93–144, Aug. 1993.

[21] Motorola, *Power PC 601 RISC Microprocessor User's Manual,* Motorola Inc, 1993.

[22] M. L. Powell, S. R. Kleinman, S. Barton, D. Shah, D. Stein, M. Weeks, *Sun multi-thread architecture,* USENIX, Winter 1991.

[23] J. Mark Stevenson, D. P. Julin, *Client-Server interactions in multi-server operating systems: The Mach-US approach*, CMU–CS–94–191, Sept. 1994.