

CS443 Paper Review  
Lei Yang  
2005-4-11

\*\*\*\*\*

Title

Lightweight Remote Procedure Call

Author

Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy

Summary

This paper presents LRPC (Lightweight Remote Procedure Call), a lightweight communication facility that combines elements of capability and RPC systems to handle communication between protection domains on a single machine.

Most important ideas

Small-kernel operating systems have borrowed the large-grained protection and programming models of RPC used in distributed computing environments and have demonstrated these to be appropriate for managing subsystems. However, they also adopt the control transfer and communication models of distributed systems. This paper pointed out that this approach is not right. They show with experiments that the common case for communication is between domains on the same machine as opposed to across machines, and that most communication is simple, involving few arguments and little data, since complex data is often hidden behind abstractions. Therefore, they conclude that although RPC is robust enough to handle both local and remote calls, conventional approaches have high over-head. On the other hand, capability systems consist of fine-grained objects sharing an address space but with their own protection domains. These offer flexibility, modularity, and protection. LRPC thus combines the control transfer and communication model of capability systems with the programming semantics and large-grained protection model of RPC. The key attributes of LRPC is: Simple control transfer, simple data transfer, simple stubs, and design for concurrency. Conventional RPC involves multiple threads that must send signals and cause context switch. In LRPC, the kernel changes the address space for the caller's thread and lets it continue to run in the server's domain. Shared buffers are pre-allocated (when the caller imports the LRPC module) for the communication of arguments and results. The caller copies the data onto the A-stack, after which no other copies are required. RPC has general stubs, but many of its features are infrequently needed. LRPC uses a stub generator that produces simple stubs in assembly language. Idle processors in multi-processor machines cache domain contexts to reduce context-switch overhead. Counters are used to keep the highest activity LRPC domains active on idle processors.

LRPC was implemented and integrated into Taos, the operating system for the DEC SRC Firefly multiprocessor workstation. Experimental results show that LRPC achieves a factor of three performance improvement over traditional approaches. Additional speedup for multiprocessor machines was also demonstrated.

Questions and doubts

I am kind of not so convinced by the claim that most communication traffic in OS is between domains on the same machine rather than between domains located on separate machines. Is the network traffic fully considered? What are the benchmarks used to evaluate the ratio? Are they representative enough?

They examined three operating systems to determine the relative frequency of cross-machine activity. Is LRPC evaluated on any of the other two OS besides Taos? What is the portability of LRPC to other systems, in terms of their interface independency of languages and platforms?

Potential Future Research

This work showed that by simplifying every aspect of an RPC -- control transfer, data transfer, linkage, and stubs, it was possible to achieve a factor of three performance improvement over the best traditional approaches based on independent threads exchanging messages. Based on the results, I think their contributions were valuable. I did a search and also found that the techniques demonstrated in LRPC are now in use Windows NT.