COMP9242: Analysis of Paper 2

Direct Inter-Process Communication (dIPC): Repurposing the CODOMs Architecture to Accelerate IPC

z5012384

November 8, 2017

Summary

The paper presents an extension known as direct IPC (dIPC) to an existing architecture known as CODOM, that provides efficient inter-process communication. CODOM has been extended and re-purposed in the implementation of dIPC to allow threads to cross process boundaries. It does so by mapping processes into a shared address space and thereby eliminating the OS from the critical path of inter-process communication.

What dIPC allows for is the ability for dIPC-enabled threads to perform regular function calls across separate processes - removing the overhead that is associated with calling into the kernel. dIPC is claimed to offer the same peformance when calling a function of another process as if the two processes were a single composite application, and does so without compromising the isolation of each.

The CODOM architecture was designed to isolate software components within a single process while providing inter-component functions calls with negligible overhead. dIPC builds on this to isolate multiple processes on a shared page table to provide secure inter-process function calls with the same negligible overhead.

The implementation of dIPC is compared with that of regular RPC calls in Linux and IPC calls of the L4 microkernel. The authors state that dIPC enables a speed increase, when communicating between processes, of 64.12 and 8.87 times respectively.

The ability to call functions within other dIPC-enabled processes is done through predefined *entry points* which are defined in the source prior to compilation. On the first call of a remote entry point of another process, a function-specific *trusted proxy* is spawned. This proxy has access to each of the caller and callee processes, and bridges calls between them. This trusted proxy performs an in-place *domain* or process switch, tracking when and where cross-domain calls and returns are executed.

Pros

The paper outlines the comparatively significant overhead in making a local RPC call, which includes context switching; page table switching; IO wait; kernel code execution; syscall dispatch trampoline; and finally syscall and userland code execution.

Cons

It is mentioned that the provided compiler allows programmers to build isolation policies for their software - using PHP as an example stating that it does not need resource isolation with the web server, nor does it need state isolation when interacting with the database. The paper states that removing such isolations eliminates *unnecessary* register and stack fiddling, further increasing IPC performance. However unsecured PHP applications on the web are a common system access point for attackers, and by removing resource isolation between the PHP interpreter and the web server; a maliciously controlled PHP process would be able to access the memory of the web server, potentially leaking confidential information.

Criticisms

The paper makes a comparison between the operation latency of a baseline Linux system running a OLTP web application stack, and that of an "ideal" *single-process* system running the same stack. No details of this single-process system are provided making it hard for the reader to determine how useful the results presented actually are. It is confusing to the reader whether such a system is desirable for actual use, besides that comment that it would be *unsafe*, hence putting the weight of such a comparison into question.

The paper makes forward references to ideas and methodologies explained later in the paper, without providing a brief description to allow the reader to continue with a basic understanding. An example of this is in section 2.2, where a forward reference is made to section 7.2 to describe the methodology used to obtain the results of the one-byte argument synchronous-IPC performance test.

Benchmark Results

References

[1] L. Vilanova, M. Ben-Yehuda, N. Navarro, Y. Etsion, and M. Valero. Codoms: Protecting software with code-centric memory domains. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 469–480, June 2014.

[2] Lluís Vilanova, Marc Jordà, Nacho Navarro, Yoav Etsion, and Mateo Valero. Direct inter-process communication (dipc): Repurposing the codoms architecture to accelerate ipc. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 16–31, New York, NY, USA, 2017. ACM.