`lua_syscall`: Specializing Operating System Kernels by Using the Lua Language

Ake Koomsin
University of Tsukuba
ake@softlab.cs.tsukuba.ac.jp

Yasushi Shinjo
University of Tsukuba
yas@cs.tsukuba.ac.jp

1. INTRODUCTION

An I/O-based application relies heavily on system calls. It would be beneficial if such an application could reduce the number of kernel context switches. In current popular operating systems, however, such a reduction is not possible. If an operating system provides a high-level interface, specialization to meet the application needs should not be difficult and will result in an improved performance compared to the use of a standard interface. We propose system call scripting that allows developers to create their own system calls based on existing system calls. This extends the concept of kernel scripting [1]. The objective of this research is to allow existing operating systems to provide application-specific specialized services.

2. APPROACH

For kernel service specialization, we use the Lua language with the LuaJIT compiler. Through this facility, we can provide the ability for developers to create their own system calls based on existing system calls. This allows executing multiple system calls in a single context switch. For this, we propose two system calls, `register_lua_syscall()` and `lua_syscall()`. The idea behind these system calls is to encapsulate a Lua state into the file descriptor. The first system call is used to create a Lua state, register the necessary bindings, load the supplied script into the Lua state, and return a file descriptor. The next system call takes a file descriptor to the Lua state and executes it with the given arguments. Because the Lua state is represented by a file descriptor, we can use the existing `close()` system call for the cleaning up operation.

3. EXPERIMENTS AND RESULTS

To evalueate our idea, we modified Memcached to use the proposed system call scripting to support UDP batch receiving and sending. We conducted two experiments to measure the average response time and number of transactions per second of GET operations. We used Memaslap for benchmarking through a Facebook test (SET operations over TCP, and multi-key GET operations over UDP), and modified it to support the Facebook test using single-key GET operations. In our experiments, both the original Memcached and our Memcached were configured to run with a single worker thread on a single CPU core and 2 GB of maximum memory to use for object storage. Both ran on the 32-bit version of FreeBSD 10.1. We configured our Memcached to process 16 messages at a time at most. Memaslap was run on 4 threads with 128 concurrencies on another machine running 64-bit Ubuntu 14.04.

Figure 1 shows the experimental results. Through our system call scripting, we found that we could reduce the average response time by 33% and achieve 43% more throughput when the value was small.

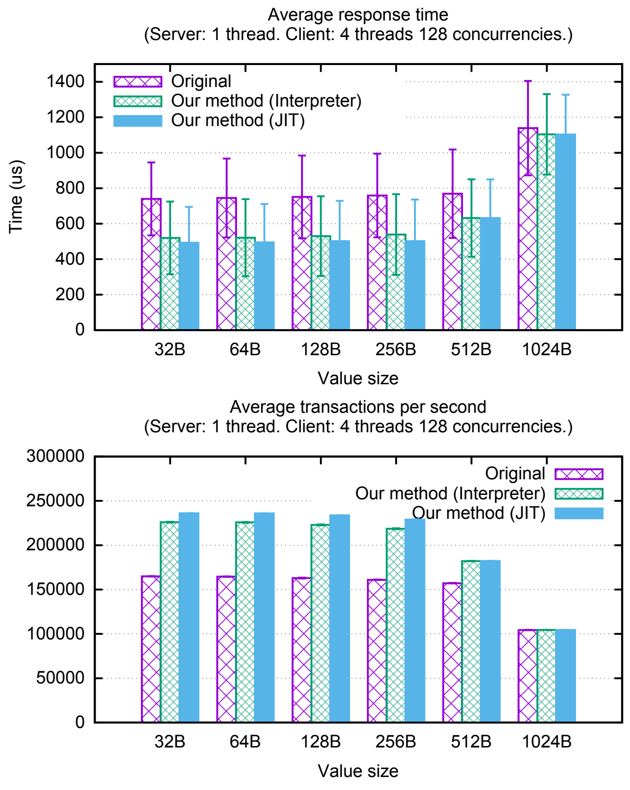

Figure 1: Experimental results

4. CONCLUSION

We proposed system call scripting that allows developers to create their own application-specific system calls based on existing system calls. This allows executing multiple system calls in a single context switch. We conducted experiments on Memcached and found a 33% reduction in the average response time and a 43% greater throughput when the value was small.

References

[1] Lourival Vieira Neto, Roberto Ierusalimschy, Ana Lúcia de Moura, and Marc Balmer. “Scriptable Operating Systems with Lua”. In: *Proceedings of the 10th ACM Symposium on Dynamic Languages (DLS ’14)*. 2014, pp. 2–10.