Vectorized System Calls

Saksham Jain   Yihe Tang   Haoran Wang
{sakshamj, yihet, haoranw2}@andrew.cmu.edu

Abstract

In this project, we introduce new vectorized system calls, vec_accept4() and vec_open(), which process multiple similar requests in kernel in a batch manner to gain speedup under heavy load by eliminating redundant work. We are able to get up to 30% improvement in both cases compared to accept4() and open(), with a few lines of code changes in Linux. We also apply vec_accept4() to speedup the connections accepted per second by Memcached server by a factor of 2, without increasing CPU utilization. Overall, we reach our "100%" goal of the project, which is to vectorize multiple system calls and gain performance improvement in an application through them.

1. Introduction

Most of the system calls are single synchronous system calls today in Linux (apart from asynchronous IO system calls, which have not been adopted widely due to the complexity in using asynchronous mechanisms [1]). This userspace-kernel interface potentially limits the throughput for applications that issue multiple system calls. Though applications in these scenarios can fork out multiple threads, each issuing separate system calls, allowing progress to be made in parallel, this doesn't address the issue of low efficiency by this method since each of those threads are potentially doing redundant work. Also, due to locks, each system call may not be progressing in parallel.

We can think of broadly three major methods to gain performance improvements for system calls on a modern CPU:

1) Use of SIMD instructions: Hardware supported parallelism can be used to process multiple data concurrently to achieve 4-8x theoretical speedup [7]. But Linux currently doesn't support SIMD (probably the overhead of saving/restoring these registers has a larger cost than improvement they can give [8]) and since system calls don't do intensive numerical computations, we don't expect much speedup in system calls by using SIMD.
2) Reduction in work: This is the approach we took to speedup system calls. There is a lot of work in a system call that is necessary if executing one system call at a time, but which is redundant if doing them in a batched fashion (e.g. mode switch from userspace to kernel-land to userspace, taking/releasing locks, doing initial setup).
3) Using multiple threads: Though it might not be a good idea to execute system calls in parallel in userspace due to possible lock contention inside kernel making them serial, it might be feasible to use threads inside vectorized system call code path to parallelize work. We leave this for future work.