Operating Systems Project 1

Measuring the overhead of system calls and context switching on Linux

Kevin Orr

Jan 30, 2018

This project attempts to measure how much overhead is introduced by executing a system call and context switching between user processes on Linux. Understanding this will allow for the student to better understand where overhead is introduced into their runtime execution, and how to take necessary steps to make code more efficient.

1 System Calls

Measuring the overhead of making a system call is simple: simply make a system call that doesn't need to do anything, rinse, and repeat. For this exercise, I followed the hint of calling `read(2)` with an argument of 0 bytes. It would be simple enough to call `read(somefd, NULL, 0)`, but when stepping through the libc implementation of `read`, I noticed there was much overhead in userland code. I wanted there to be as little overhead in user mode as possible, so I decided to use inline assembly to create a bare minimum system call.

For example on Linux on x86-64, the following assembly executes a call to `read(fd, buf, bytes)`:

```
mov rax, 0
mov rbx, fd
mov rcx, buf
mov rdx, bytes
syscall
```

Note that system call 0 is `sys_read`. So in just five instructions, we can execute a system call.

This is repeated 10 million times, so that fluctuations in cpu load won't affect timing as much, and also constant inaccuracies introduced by the system call to `gettimeofday(2)` are minimized as well.

2 Context Switching

For this half of the project, I implemented the read/write parent/child loop hinted at in the instructions. I create a bidirectional pair of pipes, fork, and then read from one pipe in one process while writing to it in the other, and then write into the other pipe while reading it in the other process. In pseudocode:

```
void parent() {
    char buf;
    for (int i=0; i<n; i++) {
        write(parent_out, &buf, 1);
        read(parent_in, &buf, 1);
    }
}

void child() {
    char buf;
    for (int i=0; i<n; i++) {
        read(child_in, &buf, 1);
        write(child_out, &buf, 1);
    }
}
```

Note that read/write is swapped in parent and child. If we instead read in both processes simultaneously, we would deadlock.

In order for this code to truly test context switching, we must make sure that parent and child can only run on the same processor. I chose to run the test once on each processor, so that if there were one process running on the same machine whose affinity is set to just one processor, it won't affect the results as a whole.

3 Results and Conclusion

On my laptop, if not much else is running, 10 million system calls takes about 3.2 seconds; 2 million context switches takes about 1.1 seconds. This is remarkably fast. Considering my CPU (i7-6500U) goes up to 3.1GHz, this means that the overhead of a system call is about 1000 cycles (~320 μs), and the overhead of context switching is about 1700 cycles (~550 μs).