EECS 678 - Lab 10

Theodore Lindsey

April 24, 2015

`http://people.eecs.ku.edu/~vvivekan/lab/mmio/mmio.html`

1. *The time required to copy the file using* `read_write` *varies with the size of the buffer specified. Smaller buffer sizes take longer. The time required for memmap varies much less regardless of how you perform the copy. Discuss why this is, and in your discussion consider the influence of: (1) the overhead of the* `read()` *and* `write()` *system calls and (2) the number of times the data is copied under the two methods.*

   Both `read` and `write` need to first read from disk, copy to a kernel buffer, then copy to a memory location in userspace, then copy back to a kernel buffer and then finally write to disk. On top of this, it repeats this over and over (especially for small buffer sizes). When we use `mmap`, we bypass most of this overhead of read-write cycles between disk, kernel buffer and userspace memory and go directly from disk to kernel buffer to disk. We also don't worry about the size of the middleman buffer and let the operating system worry about that so the number of cycles is much smaller.

2. *When you use the* `read_write` *command as supplied, the size of the copied file varies with the size of the buffer specified. When you use the memmap command implemented the size of the source and destination files will match exactly. This is because there is a mistake in the* `read_write` *code. What is the mistake, and how can it be corrected?*

   As it is written, `read_write` assume that the size of the file is a multiple of the buffer size and so it creates a destination file that is a multiple of the buffer size. The `read` command can report the number of bytes read. If instead of assuming we read in a number of bytes matching the buffer size, we track the number of bytes read and only write that many bytes, we can get around the changing filesize problem.