Chinmay Ratnaparkhi
Lab Assignment 12
EECS 678 - Introduction to Operating Systems
April 27th, 2015

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.**

   The assumption that the mmap solution would perform better than read/write solution was verified in this lab. The read/write solution completes the task by copying the contents of the desired file which is stored in a kernel-space buffer of memory, to a user-space buffer before actually copying these contents back to a kernel-space buffer that is the final destination or the location of the copy file. The mmap solution on the other hand does not have to copy the contents of the kernel-space buffer of the source into a user-space buffer before putting it into the kernel-space buffer of the destination file. It skips the intermediate copy and directly maps the kernel-space buffer to the virtual address space of the user level process. This accounts into the read/write solution requiring a little more time than mmap.

   Another reason why read/write solution is outperformed by the mmap solution is because of the buffer size limitation applied to the read/write solution. When a very small buffer size is allocated, the read/write solution has to perform a lot of iterations to copy small amounts of data per iteration and a huge overhead is generated. As the allocated buffer size goes on increasing the difference between the time taken to copy files over by the solutions goes on reducing. This is because the read/write solution does not have to iterate as many times with a large buffer size as it does with a very small buffer size, increasing its efficiency.

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?**

   With the read/write solution, when the data has to be copied a buffer size is specified and a loop is used to copy small amounts of data equal to the buffer size in

every iteration. When the file size is not an exact multiple of the buffer size, in the last iteration, the remaining amount of data to be copied is smaller than the buffer size, yet the amount copied in the last cycle is still equal to the buffer size. This makes the destination file a little bigger in size than the source file. This can be easily corrected by adding an additional step in the read/write copier where a check is performed to see if the remaining amount of data is smaller than the buffer size. If that is the case, a separate buffer variable can be used to copy the smaller amount of data to the destination. This way both of the files will have the exact same size.