Ivan Prskalo
CS351 - Systems Programming
Lab 4

Network I/O

method	**num_ops**	**+ ops/sec**	**- ops/sec**	*** ops/sec**	**/ ops/sec**
Function	1 Billion	64,050,756	64,107,676	64,029,451	63,812,169
Pipe	1 Million	1,065,489	1,215,455	1,127,426	1,106,982
Socket	1 Million	55,523	51,567	55,613	52,407

The fastest method was very clearly the function calls. It was able to outperform both pipe and socket by a considerable margin. Pipe was also considerably faster than socket, but didn't come anywhere close to the speed of function calls. Looking at the actual different operations, it seems as though the type of operation does not play a significant role in determining operations per second. The best and worst speeds by operation type were spread out among the different methods.

Some possible explanations for why function was so much faster than the other methods is because function calls are directly executed by the code. In this case, since we are in C, the code is converted into assembly language operations by the compiler and therefore runs extremely efficiently. Pipe on the other hand has overhead due to forking another process. It is essentially doing exactly what function does, but instead it runs all the operations in the child process. We know from the last lab that concurrency doesn't necessarily mean greater efficiency. It is exemplified here by the loss of speed resulting from forking a process and communicating with it for each operation. Lastly, socket is the slowest method since it has to communicate with another program all together. It isn't just another process created by the same program. It establishes a connection between client and server using TCP/IP and then remotely communicating for each operation. This is much slower than a parent process communicating with its child process through a pipeline.