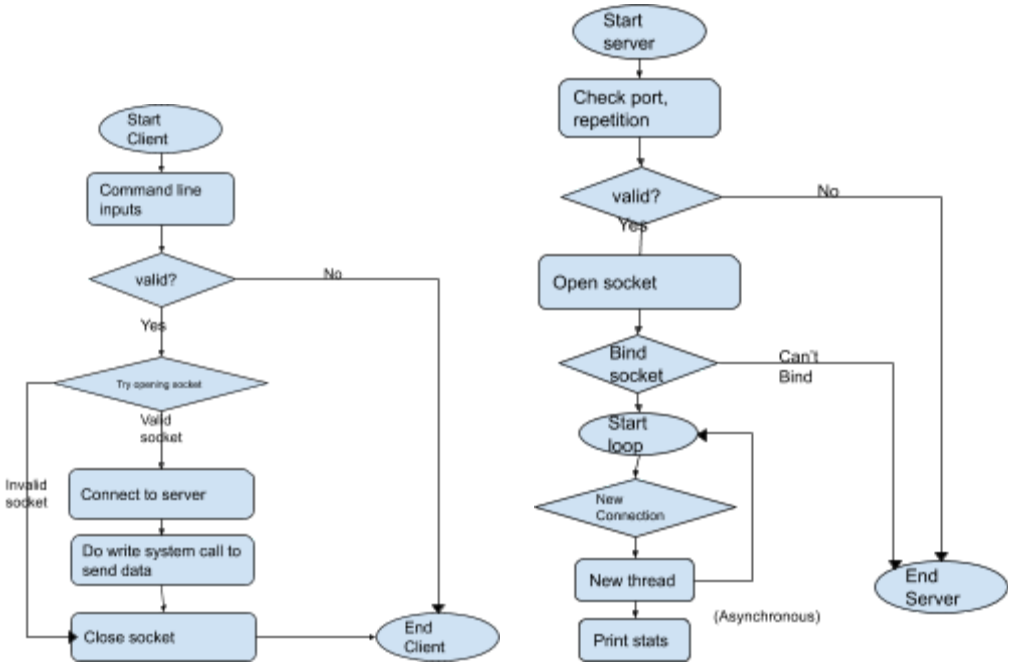
Brandon Hu
Temesgen Habte
CSS 432 Autumn 2020

# HW1 - Intro to Network Programming

## Documentation




Description:

- Client

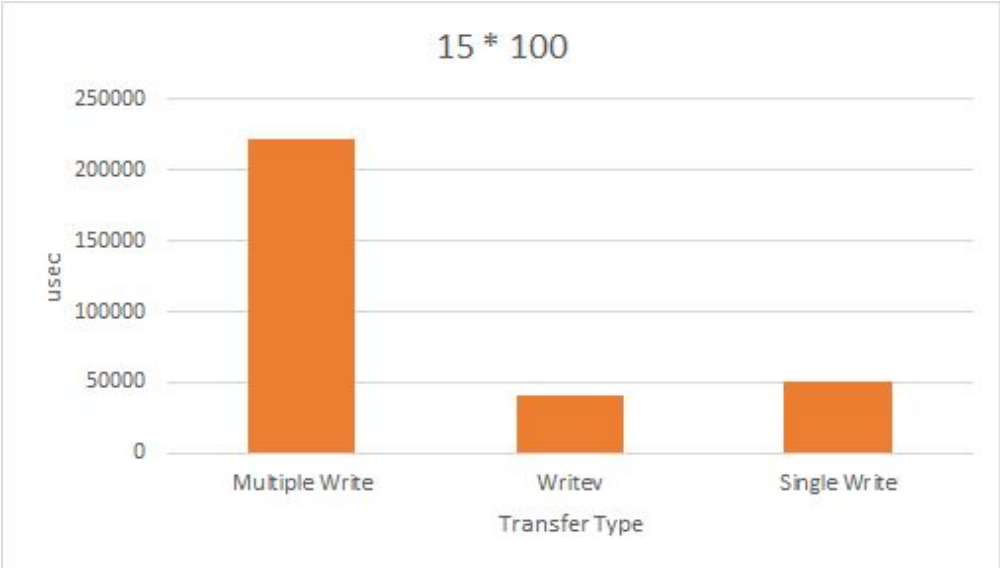
  The client program will take inputs from the command line and use that data to set the variables such as port number and type. Then it will attempt to open the socket, and afterwards connect to the server. Depending on the type specification, the program will do either multiple writes, a writev or a single write system call. Once that is completed, the client calculates and prints the send time, round trip time, and number of reads. The socket is then closed and the program is ended

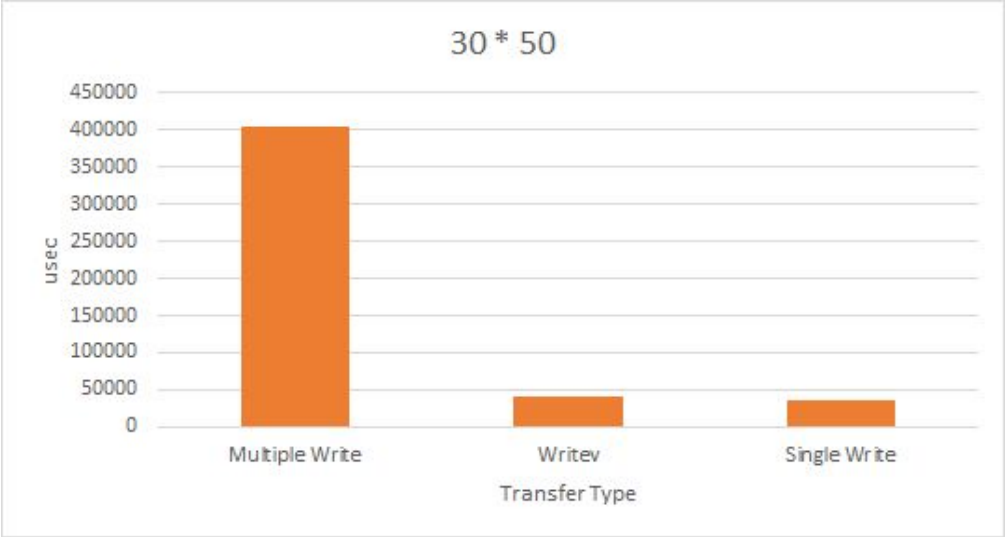
- Server

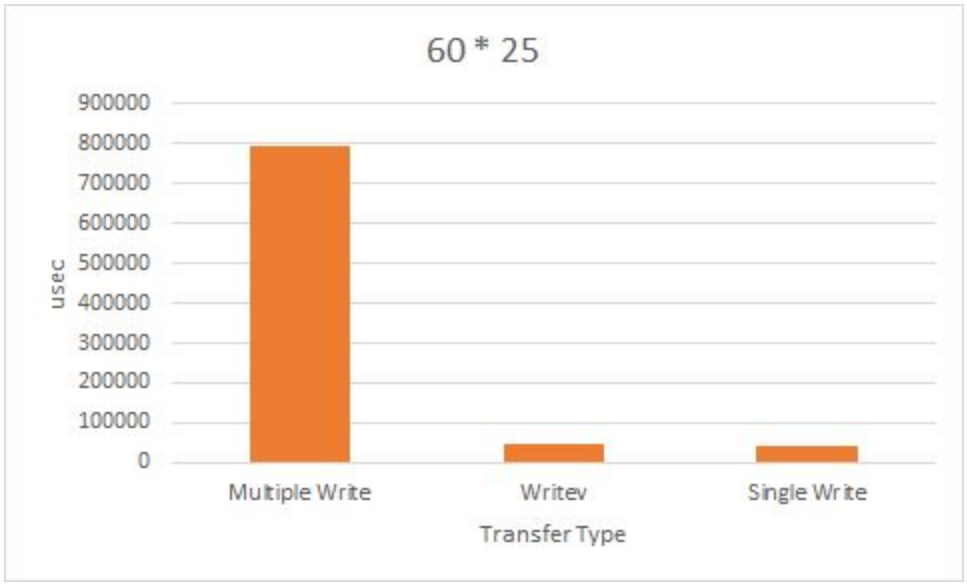
  The server program will first check the validity of the specified port number, and repetition value. Then it will open the socket and try to bind the socket to its local address. If the socket is bound, then it will start an indefinite loop that will listen for new connections to the server. When a new connection is established, a new thread is created and following this, the statistics are calculated and printed. This loop will continue until the program is closed manually. If it is not manually closed, it will be able to listen for up to 10 connections at a time.

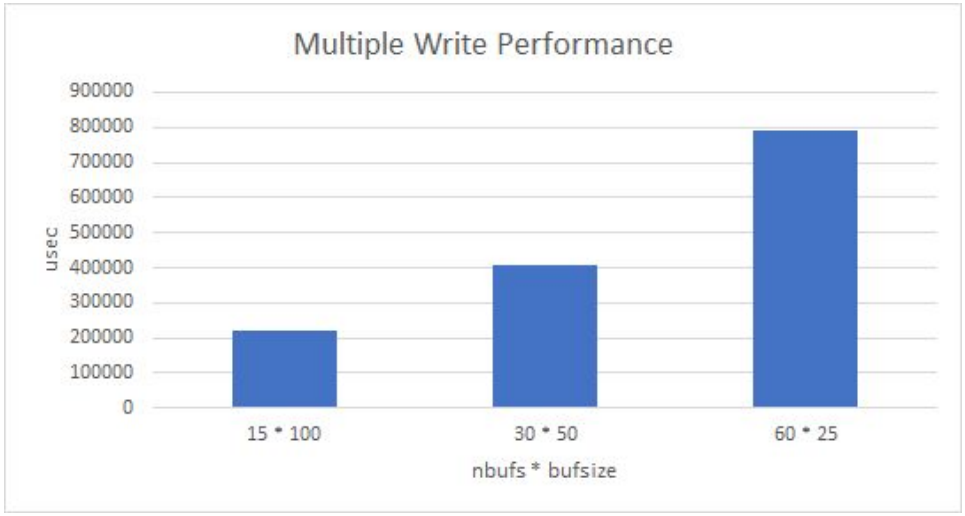
## Performance Evaluation

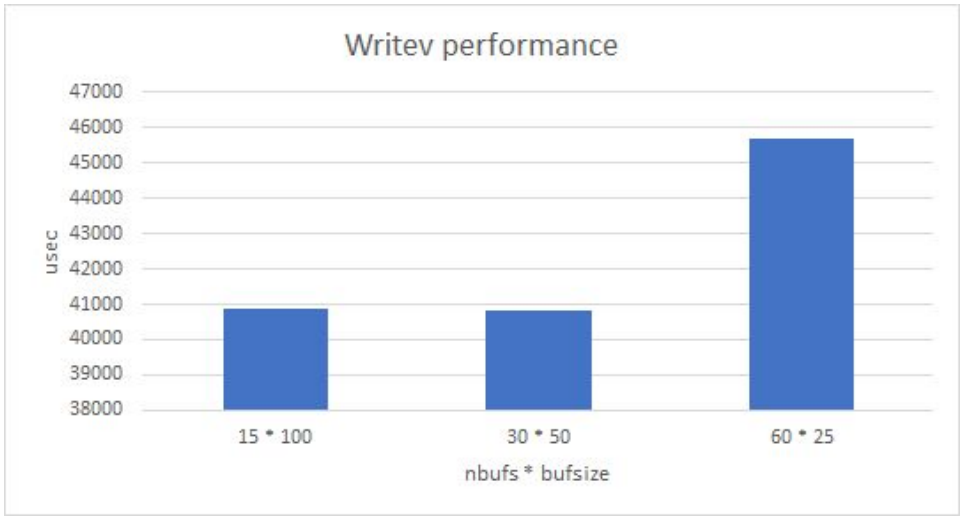
Lower is better

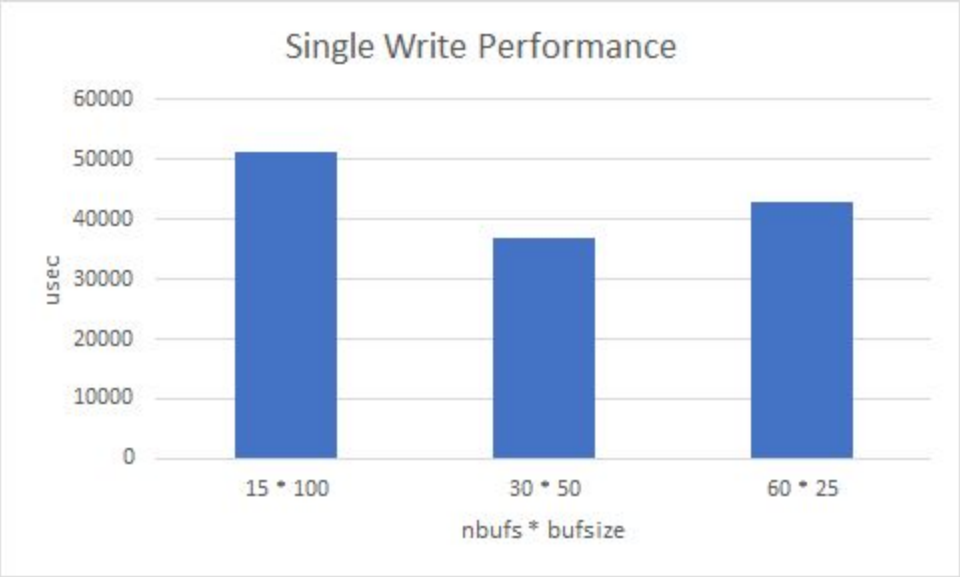



60 * 25
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
usec
Multiple Write
Writev
Single Write
Transfer Type


Multiple Write Performance
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
usec
15 * 100
30 * 50
60 * 25
nbufs * bufsize


Writev performance
47000
46000
45000
44000
43000
42000
41000
40000
39000
38000
usec
15 * 100
30 * 50
60 * 25
nbufs * bufsize




## Discussion

What we can see with the graphs above is that multiple writes consistently performs worse than writev and single write. In fact, there is a huge performance loss when using multiple writes. With 30 buffers and a buffer size of 50 bytes, multiple writes had a round trip time of about 400,000 microseconds. Meanwhile, writev and single write had a round trip time of around 40,000 microseconds. This makes sense because multiple writes calls write for every buffer unlike writev and single write where they allocate an array and send many buffers at once.

Regarding the performance of different size buffers, we can see some variation with lower and higher number of buffers. For multiple write and writev, a lower number of bigger buffers leads to greater performance. However, with single write, the performance actually was worse with the lower number of buffers. We are unsure why this is the case. We thought it would be the same as writev since they both send the whole buffer.

If we ran these tests with a slower connection, the performance would degrade as we increase the buffer size. There would be congestion and packet loss for the larger buffers. If we have more buffers and smaller buffer size, the performance would still be impacted but it would be acceptable since the packet loss problem will be resolved.

We would want to use a thread to service the connections because it will allow us to make multiple connections to the server. Having it in the main method would make it so that there could only be one connection and then the program would terminate. Even if we used a while loop, it would only allow for one connection at a time which would be useless for an application that needs multiple connections such as a game. Threads allow us to connect to multiple clients asynchronously.