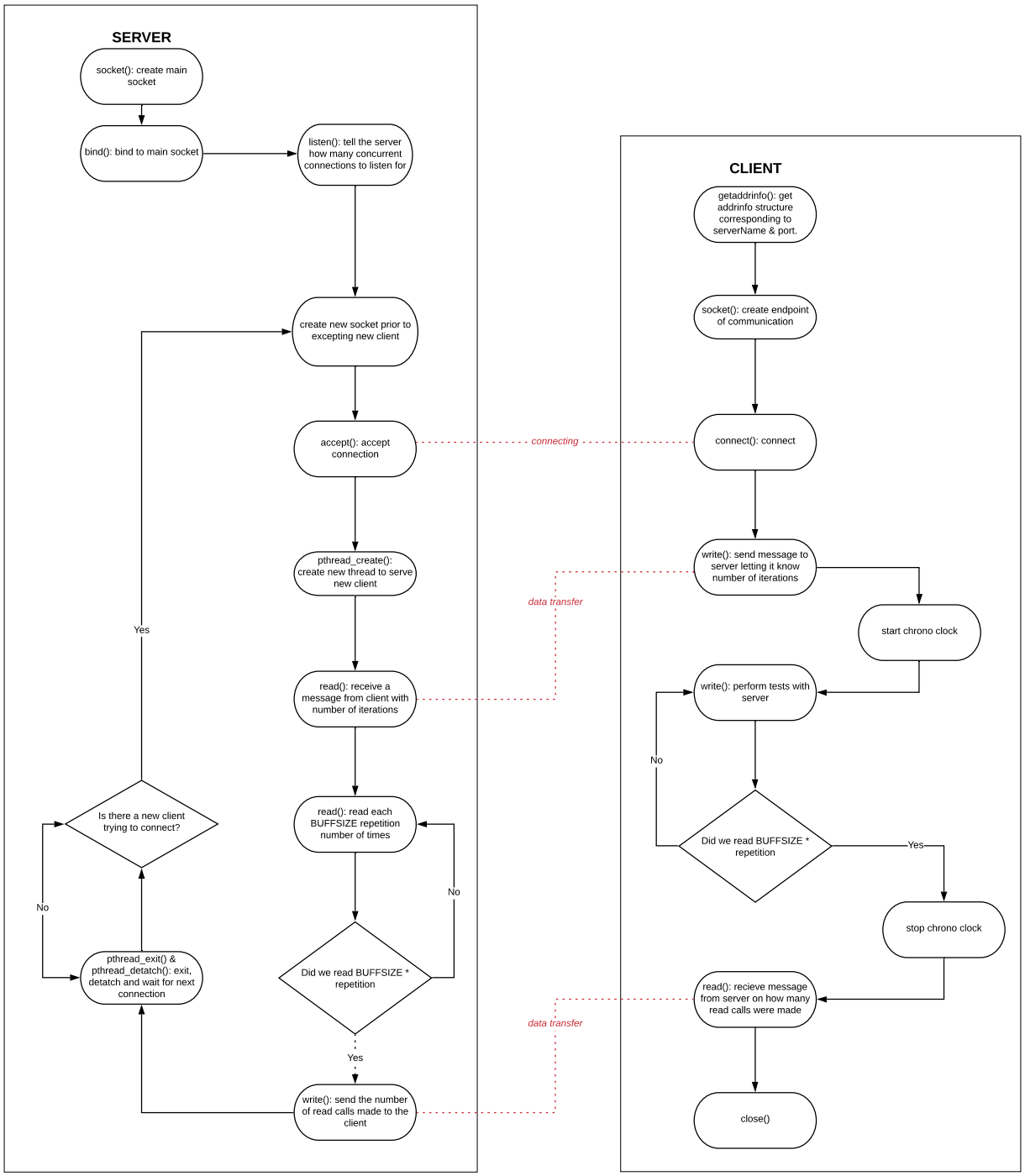
Erika Parayno
June 4, 2020
CSS 503
Program 4

Source Code, **Build Script**, & **Execution output**: See zip file
Documentation: of your implementation including explanations and illustration in one or two pages.



server.cpp implementation:

- *Reading from the client:* In this part of the code, I am filling up the buffer with size of buffsize. However, as mentioned in the program requirements documentation, a read system call may return without reading the entire data buffer. Because of this, we have to create a while loop while in the for loop of iterations. It will continue to read until it has read the buff size amount of data before it moves on to the next iteration.

- *Thread termination***:** After terminating a thread it will be detached. When this happens, its resources are automatically released back to the system without the need for another thread to join with the terminated thread.

- *Endless while loop:* As instructed, the while loop will continue until the user kills it. For each iteration in the while loop, it will create a new socket and thread to service the request.

- *Thread creation:* For this program, I implemented what we learned and the code we used from the Sleeping Barbers assignment to create pthreads. I had to create a parameter class to hold the new socket descriptor and the new data buffer. In addition to the parameter class, I also created a thread runner, which performs what needs to be done in that specific thread client, which is steps 2 to 6 from the program requirements documentation.

client.cpp implementation:

- *Conditional statements:* It is important to consider the speed and performance when performing conditional checks with the different test types because this is the only thing that the chrono timer is recording. While switch cases are commonly recognized as a faster if-else statement, it actually does not affect the speed if it is only a few test cases. Switch cases are recommended for five or more cases. In this case, since we are using less than that, I used if statements. In addition, if-else is actually better for Boolean values, which is what we are doing in this part of code.

- *Chrono type:* Since we are measuring duration, I implemented the steady_clock chrono type. This is because steady_clock represents clocks for which values of time never decreases as physical time advances, compared to the most commonly known chrono type, system_clock. Although it is unlikely that the program will be tested during daylight savings time, steady_clock remains unfettered during such an event.

Performance evaluation: showing round trip times and number of reads in an understandable manner. Also clearly show throughput.

Combination 1: bufs = 15, bufsize = 100			
Test type	Time (usec)	# All attempted reads	Throughput (Gbps)
1	308928	22721	0.77688
2	50290.1	20224	4.77231
3	44475.8	20159	5.39619

Combination 2: bufs = 30, bufsize = 50			
Test type	Time (usec)	# All attempted reads	Throughput (Gbps)
1	597746	24815	0.401509
2	74649.4	21152	3.21503
3	42110.4	20171	5.6993

Combination 3: bufs = 60, bufsize = 25			
Test type	Time (usec)	# All attempted reads	Throughput (Gbps)
1	1.18013e+06	29298	0.203367
2	124246	22012	1.93165
3	43078.5	20194	5.57122

Combination 4: bufs = 100, bufsize = 15			
Test type	Time (usec)	# All attempted reads	Throughput (Gbps)
1	1.93504e+06	33820	0.124028
2	185317	21739	1.29508
3	44234.6	20161	5.42562

Discussion:

1. Comparing your actual throughputs to the underlying bandwidth

   As discussed on 06/05/20 office hours, there was a speculation that the underlying bandwidth of the CSS labs was 5 Gbps. My results make sense as my fastest test type 3 throughputs are averaged at around 5.5 for all four combinations, like many of my peers. The bandwidth is a theoretical measurement so although my fastest throughput is slightly above the 5 gbps, it is still acceptable. Unfortunately, when trying to measure the underlying bandwidth, the CSS labs tell me that I don't have sudo permission to install tools such as iPerf or vnstat.

2. Comparisons of the performance of multi-writes, writev, and single-write performance

	Test 1	Test 2	Test 3

Average time	1005461.0	27663.5	0.3764460
Average reads	108625.6	21281.8	2.8035175
Average throughput	43474.8	20171.3	5.5230825

Test 1 (Multi-writes): The multi-writes algorithm took the most time, performed the most read calls, and had the slowest throughput. This makes sense because it performs a write for each data buffer, thus resulting in calling as many writes() as the number of data buffers.

Test 2 (Writev): The writev is in between multi-writes and single-write in average time, reads, and throughput. This makes sense because although writev sends all data buffers at once, it does contain more than one data structure, as determined by the for loop creating vectors.

Test 3: (single-write): The single-write had the least time, least reads, and fastest throughput. See how the average reads for test 3 is much lower than test 1 and test 2. Along with that it is performing near the underlying bandwidth. This makes sense because we are calling the write system call once on the array.

In conclusion, by minimizing the number system calls made in the method, you are increasing performance significantly in terms of elapsed time, number reads, and throughput.

3. Comparison of the different buffer size / number buffers combination

Increasing number of buffers: According to the table of results shown above, increasing the number of buffers results in increased elapsed time. This was pretty consistent for test type 1 and test type 2. This makes sense because with increased number of buffers, we are performing more iterations in the multiple writes and writevs code.

Decreasing the buffer size: According to the table of results shown above, decreasing the buffer size results in increased number of reads in regard to test type 1. This makes sense because in the client code, we might have to call read if the buffsize is not being read. The smaller the buffer size is of the data buffer in the client, the more read iterations the server would have to perform to read the larger buffsize on the server side.