Socket Programming – Client.cpp & Server.cpp Documentation

Author: Aldan Brown
Course: CSS 503
Date: June 2, 2025

Documentation

This is a client-server application designed to measure the performance of network data transmission using different write methods. It tests how quickly a client can send data to a server and how many read operations the server must perform in response.

Client.cpp

- Connects to a server, sends a specified amount of data using one of three transmission methods, and receives back the number of read operations the server performed.
- Key Features:
    - Accepts six command-line arguments: server name, port, repetition count, number of buffers, buffer size, and type (1, 2, or 3).
    - Types of data transmission:
        - Type 1: Sends each buffer using multiple write() calls.
        - Type 2: Uses writev() for all buffers in a single call.
        - Type 3: Sends all data in a single write() call.
    - Measures and prints transfer time, number of reads by server, and throughput in Gbps.

Server.cpp

- Accepts client connections, spawns threads to handle each client, and reads incoming data based on the client's specified number of iterations.
- Key Features:
    - Listens on a given port for incoming TCP connections.
    - Each connection is handled in a new thread (pthreads).
    - For each iteration from the client, reads 1500 bytes total.
    - After reading all data, sends the total number of read operations back to the client.

Performance Evaluation Results

The following results are from the averages of five test runs each:

Time (usec)			
Scenario	**Type 1**	**Type 2**	**Type 3**
15 100	459,085	53,270	50,636
30 50	865,396	63,570	51,321
60 25	1,696,927	74,000	47,835
100 15	2,785,235	96,607	52,137

Number of Reads			
Scenario	**Type 1**	**Type 2**	**Type 3**
15 100	22,291	17,675	20,061
30 50	24,278	20,098	20,066
60 25	28,826	20,161	20,057
100 15	32,637	20,313	20,058

Throughput (Gbps)			
Scenario	**Type 1**	**Type 2**	**Type 3**
15 100	0.52	4.55	4.76
30 50	0.28	3.81	4.70
60 25	0.14	3.25	5.03
100 15	0.09	2.49	4.73

Discussion

Comparing Actual Throughputs to Underlying Bandwidth (10 Gbps)

- Type 1 (Multi-writes) achieves throughputs of 0.9% to 5.2% of the maximum bandwidth.
- Type 2 (writev) performs significantly better, roughly 24% to 46% of the max bandwidth.
- Type 3 (single write) consistently reaches near or above 47% to 50% of the available bandwidth.

If done efficiently using a low number of large buffers, Type 2 can almost match Type 3 in maximum speed. However, neither reach more than about ½ the bandwidth due to overhead and network limitations.

Performance Comparison Among Test Types

- Type 1 (Multi-writes): The throughput is the lowest among all types. Each write incurs system call overhead.
- Type 2 (writev): This method aggregates multiple buffers into a single system call, which reduces overhead significantly. Throughputs are much higher here than Type 1, often more than an order of magnitude better, showing the efficiency gains by batching writes.
- Type 3 (Single write): Single large writes show throughput comparable to or better than writev. Sending data in a single, well-sized buffer is best.

Impact of Buffer Size / Number Buffers Combinations

- Types 1 and 2 were significantly impacted by the overhead of multiple buffers, although Type 1 was poor in performance in all buffer combinations.
- Type 3 is relatively unaffected by the number and size of buffers because it sends all the data in one go, from a single memory region, minimizing syscall and memory handling overhead.

Conclusion

For multi-write applications, overhead scales linearly with number of buffers (nbufs). Additionally, each call invokes a costly syscall entry/exit, making it the least efficient method. Writev is better than multi-writes because of its singular syscall but is less efficient than a single-write because of the kernel's multiple memory lookups and non-contiguous memory access. Single write is the most efficient because the kernel can copy a contiguous block of memory in one go.