program4 documentation and report

Documentation:

In this program, I use the client-server model where a client process establishes a connection to a server, sends data or requests, and closes the connection. The server accepts the connection and creates a thread to service the request and then wait for another connection on the main thread.

Servicing the request consists of (1) reading the number of iterations the client will perform, in my code, I just tried 20000 iterations(2) reading the data sent by the client, (3) sending the number of reads which the server performed.

First, I need run as a server, letting it wait for request, the command is as below:

./server 78400

When request arrives, the main thread of server accept a new connection, and create a thread to service the request.

What the server do in detail is to: (1) allocate dataBuf[BUFSIZE], where BUFSIZE = 1500 to read in data being sent by client, (2) receive a message by the client with the number of iterations to perform, (3) read from the client the appropriate number of iterations of BUFSIZE amounts of data (4) send the number of read() calls made as an acknowledgment to the client (5)close this connection. (6) terminate the thread.

When run as a client, the comand take six arguments: serverName, port (last 5 digits of student id), repetition (the repetition of sending a set of data buffers), nbufs (the number of data buffers), bufsize (the size of each data buffer in bytes), type (the type of transfer scenario: 1, 2, or 3)

Just as I did: ./client csslab5 78400 20000 100 15 3

which means client intends to connect the server which sits as csslab5, port is 78400, It will perform 20000 iterations of test 3. For test 3, each iteration sends 100 buffers of 15 bytes.

What the client side do is to: (1) establish a connection to a server, (2) send a message to the server letting it know the number of iterations of the test it will perform, (3)perform the appropriate number of tests with the server (measure the time this takes), (4) receive from the server a message with the number read() system calls it performed, (5) print information about the test, (6)close the socket.

Diferent types of scenarios are defined as:

Type 1: Multiple writes: invokes the write() system call for each data buffer, thus resulting in calling as many write()s as the number of data buffers, (i.e., nbufs).

Type 2: writev: allocates an array of iovec data structures, each having its *iov_base field point to a different data buffer as well as storing the buffer size in its iov_len field; and thereafter calls writev() to send all data buffers at once.

Type 3: single write: allocates an nbufs-sized array of data buffers, and thereafter calls write() to send this array, (i.e., all data buffers) at once.

Performance evaluation:

Client side: sending to csslab9

```
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 15 100 1
Test (1): time = 0.312501 sec, #reads = 22978, throughput 0.715255 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 30 50 1
Test (1): time = 0.590449 sec, #reads = 24501, throughput 0.378555 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 60 25 1
Test (1): time = 1.174245 sec, #reads = 29018, throughput 0.190350 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 15 100 2
Test (2): time = 0.054032 sec, #reads = 20142, throughput 4.136785 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 30 50 2
Test (2): time = 0.068898 sec, #reads = 20264, throughput 3.244190 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 60 25 2
Test (2): time = 0.121559 sec, #reads = 22076, throughput 1.838762 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 15 100 3
Test (3): time = 0.054405 sec, #reads = 20121, throughput 4.108379 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 30 50 3
Test (3): time = 0.059285 sec, #reads = 20137, throughput 3.770195 Gbps
[zhengsq@csslab1 os4]$ ./client csslab9 78400 20000 60 25 3
Test (3): time = 0.030117 sec, #reads = 20024, throughput 7.421532 Gbps
```

Server side: csslab9

```
[zhengsq@csslab9 os4]$ ./compile.sh
[zhengsq@csslab9 os4]$ ./server 78400
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
Exit socketThread
^C
```

Discussion:

(1) comparing actual throughputs to the underlying bandwidth

Server side: csslab9

```
[zhengsq@csslab9 os4]$ nc -vvlnp 12345 >/dev/null
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::12345
Ncat: Listening on 0.0.0.0:12345
Ncat: Connection from 10.155.176.21.
Ncat: Connection from 10.155.176.21:46762.
NCAT DEBUG: Closing fd 5.
[zhengsq@csslab9 os4]$
```

Client side: sending to csslab9

```
[zhengsq@csslab1 os4]$ dd if=/dev/zero bs=160000 count=6250 | nc -v csslab9 1234
5
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 10.155.176.30:12345.
6250+0 records in
6250+0 records out
1000000000 bytes (1.0 GB) copied, 1.46051 s, 685 MB/s
Ncat: 1000000000 bytes sent, 0 bytes received in 1.46 seconds.
[zhengsq@csslab1 os4]$
```

This will send one gigabyte of data (meaning one billion bytes) and time it. For me it took 1.46 seconds. One billion bytes is 8 billion bits aka 8 gigabits, divide by 1.46 = 5.47 Gbps, which is the "*underlying bandwidth"* between two VMs (maybe on the same host).

The throughput is the rate of successful message delivery over a communication channel. Throughput can only send as much as the bandwidth will allow, and it's usually less than that. Since clients and servers are not sitting on the real network environment, they are VMs, connected on a LAN so transferring data between them doesn't need to go over the internet. Some of them might even be different VMs on the same physical box, although our Linux lab has 1Gbps network, the "bandwidth" between two lab VMs may exceed this data, this is why I got the "underlying bandwidth" as above.

However, the results from my own experiments on two physical machines on the real network, shows me throughputs are really under the bar of bandwidth. The results are as below:

The machine is on 54 Mbps Wifi, and throughputs from the screenshot are generally 20ish Mbps. Throughput can only send as much as the bandwidth will allow, and it's usually less than that. Factors like latency (delays), jitter (irregularities in the signal), and error rate (actual mistakes during transmission) can reduce the overall throughput.

(2) comparisons of the performance of multi-writes, writev, and single-write performance

Comparing three types of writing functions, we can see the one with the least throughput is the test 1, namely, multiple writes, invokes as many write()s as the number of data buffers, which leads to a bunch of overheads, test 2 and test 3 have roughly the same performance, they are both using unique data structures to store data buffer, and send it at one time, this is why they render a good performance.

(3) comparison of the different buffer size / number buffers combinations

For test 3, different combinations wouldn’t affect it at all, because it only care about their product which is a constant number.

For test 1 and test 2, we can see, with the incrementing of the number buffers(nbufs), their performance decrease, which means, in terms of performance, the times of write on TCP/IP really matter.