

HW1 - Intro to Network Programming

[Re-submit Assignment](#)

Due Apr 7 by 11:59pm **Points** 25 **Submitting** a file upload
File Types zip and tgz **Available** after Mar 27 at 12am

1. Overall Requirement

The first assignment will require you to create a multi-threaded client/server and evaluate the reads and writes made by the system. The focus of this assignment is not on the threads, so we will not be covering them in detail like you will in an Operating Systems course.

Your program should consist of two parts. Please create two files accordingly, one is for the client (e.g., client.cpp) and one is for the server (e.g., server.cpp).

- **Server.** The server will create a TCP socket that listens on a port (the last 4 digits of your ID number unless it is < 1024, in which case, add 1024 to your ID number). The server will accept an incoming connection and then create a new thread (use the pthreads library) that will handle the connection. The new thread will read all the data from the client and respond back to it (acknowledgement). The response detail will be provided in Server.cpp.
- **Client.** The client will create a new socket, connect to the server and send data using 3 different ways of writing data (data transferring). It will then wait for a response and output the response.
- Perform this task between two computers on the UW320 lab network. Any tow from **uw1-320-00.uwb.edu** though **uw1-320-15.uwb.edu**

See Beej's programming guide and the lecture slides for information on using sockets.

See <http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html> (<http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html>) or <https://computing.llnl.gov/tutorials/pthreads/> [\(https://computing.llnl.gov/tutorials/pthreads/\)](https://computing.llnl.gov/tutorials/pthreads/) for information on using pthreads.

2. Detailed Requirement

Client.cpp

Your client program must receive the following six arguments:

1. **port:** server's port number
2. **repetition:** the repetition of sending a set of data buffers

3. **nbufs**: the number of data buffers
4. **bufsize**: the size of each data buffer (in bytes)
5. **serverip**: server's IP address
6. **type**: the type of transfer scenario: 1, 2, or 3 (see below)

From the above parameters, you need to allocate data buffers below:

```
char databuf[nbufs][bufsize]; // where nbufs * bufsize = 1500
```

The three transfer scenarios are:

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**).

```
for ( int j = 0; j < nbufs; j++ )
    write( sd, databuf[j], bufsize ); // sd: socket descriptor
```

2. **writenv**: 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 `writenv()` to send all data buffers at once.

```
struct iovec vector[nbufs];
for ( int j = 0; j < nbufs; j++ ) {
    vector[j].iov_base = databuf[j];
    vector[j].iov_len = bufsize;
}
writenv( sd, vector, nbufs ); // sd: socket descriptor
```

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.

```
write( sd, databuf, nbufs * bufsize ); // sd: socket descriptor
```

The client program should execute the following sequence of code:

1. Open a new socket and establish a connection to a server.
2. Allocate **databuf[nbufs][bufsize]**.
3. Start a timer by calling **gettimeofday**.
4. Repeat the **repetition** times of data transfers, each iteration is performed based on the specified **type** such as **1: multiple writes**, **2: writenv**, or **3: single write**
5. Lap the timer by calling **gettimeofday**, where `lap - start = data-sending time`.
6. Receive from the server an integer acknowledgement that shows how many times the server called `read()`.
7. Stop the timer by calling **gettimeofday**, where `stop - start = round-trip time`.
8. Print out the statistics as shown below:

```
Test 1: data-sending time = xxx usec, round-trip time = yyy usec, #reads = zzz
```

9. Close the socket.

Server.cpp

Your server program must receive the following two arguments:

1. **port**: server's port number
2. **repetition**: the repetition of client's data transmission activities. This value should be the same as the "repetition" value used by the client.

The main function should be:

1. Accept a new connection
2. Create a new thread
3. Loop back to the accept command and wait for a new connection

The server must include **your_function** (whatever the name is) which is the function called by the new thread. This function should:

1. Allocate **databuf[BUFSIZE]**, where BUFSIZE = 1500.
2. Start a timer by calling **gettimeofday**.
3. Repeat reading data from the client into databuf[BUFSIZE]. Note that the **read** system call may return without reading the entire data if the network is slow. You have to repeat calling **read** like:

```
for ( int nRead = 0;
      ( nRead += read( sd, buf, BUFSIZE - nRead ) ) < BUFSIZE;
      ++count );
```

Check the manual page for **read** carefully.

4. Stop the timer by calling **gettimeofday**, where stop - start = data-receiving time.
5. Send the **number of read() calls made**, (i.e., **count** in the above) as an acknowledgement.
6. Print out the statistics as shown below:

```
data-receiving time = xxx usec
```

7. Close this connection.
8. Optionally, terminate the server process by calling **exit(0)**. (This might make it easier for debugging at first. In reality you would not want to do this on a multi-threaded server).

Your performance evaluation should cover the following nine test cases:

1. **repetition** = 20000
2. Three combinations of **nbufs * bufsize** = 15 * 100, 30 * 50, and 60 * 25
3. Three test scenarios such as **type** = 1, 2, and 3

You may have to repeat your performance evaluation and to average elapsed times.

3. What to submit

You must submit the following deliverables in a zip file. If your code does not work in such a way that it could have possibly generated your results, you will not receive credit for your results. Some points in the documentation, evaluation, and discussion sections will be based on the overall professionalism of the document you turn in. You should make it look like something you are giving to your boss and not just a large block of unorganized text.

Criteria	Percentage
Documentation of your algorithm including explanations and illustrations in one or two pages	10
Source code that adheres good modularization, coding style, and an appropriate amount of comments. The source code is graded in terms of (1) correct tcp socket establishment, (2) three different data-sending scenarios at the client, (3) instantiation of the thread on the server, (4) use of gettimeofday and correct performance evaluating code, and (5) comments, etc.	35
Execution output such as a snapshot of your display/windows. Or, submit partial contents of standard output redirected to a file. You don't have to print out all data. Just one page evidence is enough.	5
Performance evaluation that summarizes performance data a table.	10
Discussion should be given comparing multi-writes, writev, and single-write performance. Discuss how the results would be different (other than just being slower) if you ran this on a slower network (say, 1 Mbps). Additionally, discuss the why you want to use a thread to service the connection rather than servicing it in the main function.	40
Total	100

Your grade for this assignment will be "total points/4."

Some general help --

Client

1. Receive a server's IP port (**server_port**) and IP name (**server_name**) as Linux shell command arguments.
2. Retrieve a **hostent** structure corresponding to this IP name by calling **gethostbyname()**. Alternatively, you can use **getaddrinfo** as **gethostbyname** has been deprecated.
<http://beej.us/guide/bgnet/html/single/bgnet.html#getaddrinfo>
[_ \(http://beej.us/guide/bgnet/html/single/bgnet.html#getaddrinfo\)_](http://beej.us/guide/bgnet/html/single/bgnet.html#getaddrinfo)

```
struct hostent* host = gethostbyname( server_name );
```

3. Declare a **sockaddr_in** structure, zero-initialize it by calling **bzero**, and set its data members as follows:

```
int port = YOUR_ID; // the last 4 digits of your student id
sockaddr_in sendSockAddr;
bzero( (char*)&sendSockAddr, sizeof( sendSockAddr ) );
sendSockAddr.sin_family      = AF_INET; // Address Family Internet
sendSockAddr.sin_addr.s_addr =
    inet_addr( inet_ntoa( *(struct in_addr*)&host->h_addr_list ) );
sendSockAddr.sin_port        = htons( server_port );
```

4. Open a stream-oriented socket with the Internet address family.

```
int clientSd = socket( AF_INET, SOCK_STREAM, 0 );
```

5. Connect this socket to the server by calling **connect** as passing the following arguments: the socket descriptor, the `sockaddr_in` structure defined above, and its data size (obtained from the **sizeof** function).

```
connect( clientSd, ( sockaddr* )&sendSockAddr, sizeof( sendSockAddr ) );
```

6. Use the **write** or **writen** system call to send data.

7. Use the **read** system call to receive a response from the server.

8. Close the socket by calling **close**.

Server

1. Declare a **sockaddr_in** structure, zero-initialize it by calling **bzero**, and set its data members as follows:

```
int port = YOUR_ID; // the last 4 digits of your student id
sockaddr_in acceptSockAddr;
bzero( (char*)&acceptSockAddr, sizeof( acceptSockAddr ) );
acceptSockAddr.sin_family      = AF_INET; // Address Family Internet
acceptSockAddr.sin_addr.s_addr = htonl( INADDR_ANY );
acceptSockAddr.sin_port        = htons( port );
```

2. Open a stream-oriented socket with the Internet address family.

```
int serverSd = socket( AF_INET, SOCK_STREAM, 0 );
```

3. Set the `SO_REUSEADDR` option. (Note this option is useful to prompt OS to release the server port as soon as your server process is terminated.)

```
const int on = 1;
setsockopt( serverSd, SOL_SOCKET, SO_REUSEADDR, (char *)&on,
    sizeof( int ) );
```

4. Bind this socket to its local address by calling **bind** as passing the following arguments: the socket descriptor, the `sockaddr_in` structure defined above, and its data size.

```
bind( serverSd, ( sockaddr* )&acceptSockAddr, sizeof( acceptSockAddr ) );
```

5. Instruct the operating system to listen to up to n connection requests from clients at a time by calling **listen**.

```
listen( serverSd, n );
```

6. Receive a request from a client by calling **accept** that will return a new socket specific to this connection request.

```
sockaddr_in newSockAddr;  
socklen_t newSockAddrSize = sizeof( newSockAddr );  
int newSd = accept( serverSd, ( sockaddr *)&newSockAddr, &newSockAddrSize );
```

7. Use the **read** system call to receive data from the client. (Use newSd but not serverSd in the above code example.)
8. Use the **write** system call to send back a response to the client. (Use newSd but not serverSd in the above code example.)
9. Close the socket by calling **close**.

You need to include the following header files so as to call these OS functions:

```
#include <sys/types.h>    // socket, bind  
#include <sys/socket.h>  // socket, bind, listen, inet_ntoa  
#include <netinet/in.h>  // htonl, htons, inet_ntoa  
#include <arpa/inet.h>   // inet_ntoa  
#include <netdb.h>       // gethostbyname  
#include <unistd.h>      // read, write, close  
#include <strings.h>     // bzero  
#include <netinet/tcp.h> // SO_REUSEADDR  
#include <sys/uio.h>     // writev
```

Some Rubric

Criteria	Ratings		Pts
Source Code <i>Sockets, threading, use of correct test cases, comments</i>	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
Discussion <i>Comparison and analysis of results</i>	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
Documentation <i>Description of the algorithm(s)</i>	2.0 pts Full Marks	0.0 pts No Marks	2.0 pts
Performance <i>Table of performance data</i>	2.0 pts Full Marks	0.0 pts No Marks	2.0 pts
Output <i>Example output</i>	1.0 pts Full Marks	0.0 pts No Marks	1.0 pts
Total Points: 25.0			