CSE 589 Fall 2013

Programming Assignment 1

Distributed File Sharing System

Due Time : 10/03/2013 @ 23:59:59

---

1. Objective

Getting Started: Familiarize yourself with socket programming.

Implement: Develop a simple application for distributed file sharing among remote hosts and observe some
network characteristics using it.

Analyze: Understand the packet-switching network behavior and compare the results of your file sharing application for measuring network performance.

2. Getting Started

2.1 Socket Programming
- Beej Socket Guide: http://beej.us/guide/bgnet

3. Implementation

3.1 Programming environment

You will write C (or C++) code that compiles under the GCC (GNU Compiler Collection) environment. If you use any other language apart from C or C++ your project will not be graded.

Furthermore, you should ensure that your code compiles and operates correctly on the CSE student servers . This means that your code should properly compile by using the version of g++ (for C++ code) or gcc (for C code) found on the CSE student servers and should function correctly when executed.

3.2 Running your program

Your process (your program when it is running in memory) will take 2 command line parameters. The first parameter indicates whether your program instance should run as a server or a client. The second parameter corresponds to the port on which your process will listen for incoming connections. (e.g., if your program is called prog1, then you can run it like this: ./prog1 s 4322, where the “s” indicates that it is the server and 4322 is the port. Suppose you want to run it as a client then you should run it as ./prog1 c 4322 where the “c” parameter indicates that the process is a client and 4322 is the listening port.)

NOTE: Servers should always be run on timberlake and no other host involved in file transfer should be run on timberlake. Use other CSE student servers to run hosts involved in file transfer.

NOTE: Use TCP Sockets only for your implementation. You should use only the select() API for handling multiple socket connections. Please don’t use multi-threading or underground-exec.

Servers to be used:

timberlake (only to run server. Don't do any file exchange on timberlake).

Following servers can be used for file-exchange.

Euston
embankment
underground
highgate

NOTE: You should use only the servers listed above. Don't use any other department servers.

Also, create your own directory in /local/Fall 2013/ directory on all the hosts except timberlake where you can use you home directory to store your programs/files on the servers.

Use only this directory to store/run your programs. Also, make sure you have set appropriate permissions to your folder so that it cannot be accessed by any other students.

NOTE: Please don't use any other directory/cse servers to run your code.

3.3 Functionality of your program

When launched, your process should work like a UNIX shell. It should accept incoming connections and at the same time provide a user interface that will offer the following command options: (Note that specific examples are used for clarity.)

1. HELP Display information about the available user interface options.

2. MYIP Display the IP address of this process.

Note: The IP should not be your “Lo” address (127.0.0.1). It should be the actual IP of the host.

3. MYPORT Display the port on which this process is listening for incoming connections.

4. REGISTER <server IP> <port_no>: This command is used by the client to register itself with the server and to get the IP and listening port numbers of all the peers currently registered with the server.

The first task of every host is to register itself with the server by sending the server a TCP message containing its listening port number. The server should maintain a list of the IP address and the listening ports of all the registered clients. Let’s call this list as “Server-IP-List”. Whenever a new host registers or a registered host exits, the server should update its Server-IP-List appropriately and then send this updated list to all the registered clients. Hosts should always listen to such updates from the server and update their own local copy of the available peers. Any such update which is received by the host should be displayed by the client. The REGISTER command takes 2 arguments. The first argument is the IP address of the server and the second argument is the listening port of the server. If the host closes the TCP connection with the server for any reason then that host should be removed from the “Server-IP-List” and the server should promptly inform all the remaining hosts.

Note: The REGISTER command works only on the client and should not work on the server.

5. CONNECT <destination> <port no>: This command establishes a new TCP connection to the specified <destination> at the specified < port no>. The <destination> can either be an IP address or a hostname. (e.g., CONNECT timberlake.cse.buffalo.edu 3456 or CONNECT 192.168.45.55 3456). The specified IP address should be a valid IP address and listed in the Server-IP-List sent to the host by the server. Any attempt to connect to an invalid IP or an IP address not listed by the server in its Server-IP-List should be rejected and suitable error message displayed. Success or failure in connections between two peers should be indicated by both the peers using suitable messages. Self-connections and duplicate connections should be flagged with suitable error messages. Every client can maintain **up-to 3 connections with its peers. Any request for more than 3 connections should be rejected.** In addition every host should always maintain a constant connection with the server.

6. LIST: Display a numbered list of all the connections this process is part of. This numbered list will include connections initiated by this process and connections initiated by other processes. The output should display the hostname, IP address and the listening port of all the peers the process is connected to. Also, this should include the server details.

E.g., id: Hostname	IP address	Port No.
1: timberlake.cse.buffalo.edu	192.168.21.20	4545
2: highgate.cse.buffalo.edu	192.168.21.21	5454
3: underground.cse.buffalo.edu	192.168.21.22	5000
4: embankment.cse.buffalo.edu	192.168.21.22	5005
5: euston.cse.buffalo.edu	192.168.21.22	5125

Note: The connection id 1 should always be your server running on timberlake.cse.buffalo.edu. The remaining connections should be the peers whom you have connected to.

7. TERMINATE <connection id.> This command will terminate the connection listed under the

specified number when LIST is used to display all connections. E.g., TERMINATE 2. In this example, the connection with highgate should end. An error message is displayed if a valid connection does not exist as number 2. If a remote machine terminates one of your connections, you should also display a message.

8. EXIT Close all connections and terminate this process. When a host exits, the server de-registers the host and sends the updated "Server-IP-List" to all the clients. Other hosts on receiving the updated list from the server should display the updated list.

9. DOWNLOAD <file_name> <file_chunk_size_in_bytes>

The host shall download in parallel different chunks of file specified in the download command from all connected hosts until the complete file is downloaded. Please note that the total number of hosts that the client can download from **can be 1, 2, or 3.** The hosts will be part of the list of hosts displayed by the LIST command.

E.g., Suppose there is a file named "mytext.txt" of size 100bytes available with hosts x,y and z.

A new client registers (register command) with the server and gets the connection details of x,y and z. It then connects to all 3 hosts (Connect command). Then the command DOWNLOAD mytext.txt 10 is entered on the client to initiate the download process.

The client then sends requests to x, y and z to download 10 bytes each of the file mytext.txt from

the three hosts. For instance, the client can send request to host x to download from 0-9 bytes then to host z to download from 10-19 bytes and then to y to download from 20-29 bytes. The client should download from all the servers in parallel and not wait for one download to complete before sending a new request.

Suppose the client completes downloading the chunk of file from 0-9 bytes from host x before it completes the download of chunks from host y and z then the client should immediately send a new request to download the next needed chunk of file (30-39) and not wait till the download completes from y and z.

Once the file is downloaded completely, the client should display the details about the chunks downloaded from each host. So, in the above scenario the client should display something like shown below:

Host	Host-id	File chunks downloaded
x	2	0-9, 30-39, 50-59, 60-69, 90-99
y	3	20-29, 40-49, 70-79
z	4	10-19,80-89

(The key idea is to understand that the download rate can vary for each host. The client cannot expect the same performance from every host it is connected to.)

We will compare the downloaded file to the shared file and if the files are different then it will be considered as an incomplete download.

NOTE: During download some hosts may crash/exit/stop responding. In this situation the client should then continue to download the remaining chunks from available servers. If the host crashes during transfer of data and the client cannot get the whole chunk of the file, the the client should discard the chunk and the request some other available host to share the discarded chunk. Client should download from all the available hosts and no connection should be left unused.

NOTE: Both notifications and data sending should use TCP connections. Make sure the clients send

their files in parallel even though they cannot start sending files exactly at the same time.

NOTE: DOWNLOAD command on a server should display an error message. No files should be

downloaded from the server.

11. CREATOR Display your (the students) full name, your UBIT name and UB email address.

4. Analyze

Make sure you test your program for some values of file sizes between 1000 Bytes and 10 MBytes,

and then for sizes {50, 70, 100, 150, 200} MBytes. You should also test your program for different packet sizes (or the size of the buffer you read from the file and send at a time using the send socket call), ranging from 100 Bytes to 1400 Bytes. These two parameters will be referred to as File Size and packet size.

You can generate files of different sizes using the UNIX utility dd. For example, to generate a file of size 512 bytes, use the command:

dd if=/dev/zero of=test_file_to_create count=1

Here count=1 refers to 1 block of 512 bytes.

NOTE: Please delete these test files once you have tested your application and do not include any of these in your submission.

4.1 Data Rates vs. File Size

Run your application for some value of file size between 1000 Bytes and 10 MBytes, and then for sizes {50, 70, 100, 150} MBytes. Observe the Tx Rate and Rx Rates. Keep the packet size to be constant at 1000 Bytes and do a single file transfer at any time for these measurements.
Write down your observations. What variations did you expect for data rates by changing the file size and why? Do they agree with your measurements; if not then why? Remember to analyze the entire system, which includes not only the network, but also the Tx and Rx ends.

4.2 Data Rates vs. Packet Size

Run your application for different packet sizes (ranging from 100 Bytes to 1400 Bytes, by increasing in steps of 200 Bytes), and observe the Tx Rate and Rx Rates. Keep the file size constant at 150 MBytes and do a single file transfer at any time for these measurements. Write down your observations. What variations did you expect for data rates by changing the packet size and why? Do they agree with your measurements; if not then why? Remember to analyze the entire system, which includes not only the network, but also the Tx and Rx ends.

4.3 Data Rates vs. Load Variations

Run your application on different machines, and start multiple file transfers over the network at the same time. Vary the load over the network, by varying the number of simultaneous file transfers from 1 through 3, i.e., {1,2,3}. Keep the File Size constant at 70 Mbytes and Packet Size at 1000 Bytes for all transfers.

Write down your observations. What variations did you expect for data rates by changing the load and why? Do they agree with your measurements; if not then why? Remember to analyze the entire system, which includes not only the network, but also the Tx and Rx ends.

4.4 Data Rates vs. Number of hosts.

Run your application on different machines, and start single file transfer for file size 150MB over the network at the same time. Vary the number of hosts sharing the file from **1, 2 and 3.** Note the time taken for download in each case and then analyze the results and conclude as to why the download time varies (if they) when the number of hosts change.

Briefly record all your analysis and observations for sections 4.1,4.2, 4.3 and 4.4 in a file called ubit_name_analysis.pdf and turn in with your implementation.

NOTE: For all the above analysis you decide on the file-chunk size. The selected file-chunk size should be meaningful and should aid your analysis. Don't forget to specify the file chunk size you have used in your analysis. Use different file chunk size and don't keep it constant for all the analysis experiments.

5. Submission and Grading

5.1 What to Submit ?

Your submission should contain a tar file – Name it as <your ubit_name>.tar:

All source files including Makefile (Name your main program as ubit_name_proj1.c)

Your analysis for sections 4.1, 4.2, 4.3 and 4.4 in a file named <ubit_name _analysis.pdf>.

(Usage of graphs for your analysis is highly recommended. Please don't write huge texts.)

5.2 How to submit

Use the submission command submit_cse589 to submit the tar file.

5.3 Grading Criteria

Correctness of output

Your analysis

Organization and documentation of your code

NOTE: A detailed grading (marks-split) for project 1 will be given later.

5.4 Important Key Points:

There is just one program. DON'T submit separate programs for client and server.

All commands are case-insensitive.

Error Handling is very important – Appropriate messages should be displayed when something goes bad.

DON'T ASSUME. If you have any doubts in project description ask the TA's.

Please try to avoid multiple submissions. If you have any multiple submissions we will consider the latest submission before the deadline.

Submission deadline is hard. No extension. Any submission after 23:59:59 of Oct 3rd 2013 will be considered as late submission.

Please do not submit any binaries or object files or any test files.