CSE 489/589 Spring 2012
Programming Assignment 1
Remote File Sharing System

Due Time 1: 02/17/2012 @ 23:59:59 (for graduates);
Due Time 2: 02/24/2012 @ 23:59:59 (for undergraduates)

1. Objective

Getting Started: Familiarize yourself with socket programming.
Implement: Develop a simple application for 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 with those of a standard tool, iperf, used for measuring network performance.

2. Getting Started

2.1 Socket Programming

Beej Socket Guide: http://beej.us/guide/bgnet

2.2 iPerf

Iperf Homepage: http://iperf.sourceforge.net/

3. Implement

3.1 Programming environment

You will write C++ (or C) code that compiles under the GCC (GNU Compiler Collection) environment. Furthermore, you should ensure that your code compiles and operates correctly on the CSE student servers (timberlake.cse.buffalo.edu, nickelback.cse.buffalo.edu, metallica.cse.buffalo.edu and dragonforce.cse.buffalo.edu). 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 1 command line parameter that 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 4322, where 4322 is the port.)

NOTE: Use TCP Sockets for your implementation.

3.3 Functionality of your program

When launched, your process will listen for incoming connections on the specified port. Meanwhile,

your process will 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. EXIT Close all connections and terminate this process.

3. MYIP Display the IP address of this process.

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

5. CONNECT Establish a new connection to the machine at the specified IP address that is listening on the specified port (e.g., CONNECT nickelback.cse.buffalo.edu 3456). After using this command, the user interface on this process will remain unavailable until a success message or failure message is displayed. Whenever a remote process tries to connect to this process, the connection (if possible) is set up automatically and a message is displayed. You should not allow more than one connection with a particular remote host. You should not allow connections to yourself. Each process should never maintain more than 5 connections.

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.
E.g., 1: nickelback.cse.buffalo.edu
2: metallica.cse.buffalo.edu
3: dragonforce.cse.buffalo.edu

7. TERMINATE This 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 metallica 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. UPLOAD For example, UPLOAD 3 ../files/a.txt. This will upload the file a.txt which is located in the local path ../files/ to the host on the connection that is designated by the number 3 when LIST is used. An error message is displayed if the file was inaccessible or if 3 does not represent a valid connection. The remote machine will automatically accept the file and save it in a different directory under the original name (Since all those servers use the same storage space, the new file received by the host should not overwrite the original file). When the upload is complete this process will display a message indicating so. Also, the remote machine will display a message in its user interface indicating that a file (called a.txt) has been downloaded. When an upload is occurring, the user interface of the uploading process will remain unavailable until the upload is complete. Upon completion, a message is displayed. If the upload fails for some reason, an error message should be displayed. When an upload is occurring, a message should be displayed on the remote machine when the upload begins. If

the upload fails for some reason, an error message should be displayed on the remote machine.

At the end of each successful upload, you will print the rate at which the Transmitter (Sender/Uploader) uploaded the file. Similarly, you will also print at the receiver, the rate at which the file was received. We will call these as Tx Rate and Rx Rate. Tx rate is the amount of data (in Bits) transferred from the Tx end, divided by the time taken by Tx to read and send the complete file in chunks of Packet Size bytes each. The Rx rate is defined similarly (in Bits/sec) as the total file size received (in bits) divided by the time taken to receive the file over the socket and write it. For example, if a file was uploaded from timberlake to nickelback, the format for printing this information is:
At Tx end:
Tx (timberlake): timberlake -> nickelback, File Size: x Bytes, Time Taken: y useconds, Tx Rate: z bits/second.
At Rx end:
Rx (nickelback): timberlake -> nickelback, File Size: a Bytes, Time Taken: b useconds, Rx Rate: c bits/second.

NOTE: You should read the file in chunks of packet size-byte buffers and send those buffers using the send socket call, instead of reading the whole file at once. You can use UNIX utility function *gettimeofday* to know the time taken for receiving and sending the file at the two ends. Make sure to appropriately call this function in your program to account only for the time taken for uploading (reading and sending) and downloading (receiving and writing) the file. After printing this information, flush the standard output using *fflush*, to immediately print this information.

9. DOWNLOAD This will download a file from each connection shown by LIST command at the same time. E.g., after the process on timberlake receives this command, it will notify nickelback, metallica and dragonforce to send their files parallel with TCP connections. The local machine will automatically accept the file and save it in a different directory under the original name for the same reason which is explained in UPLOAD. Those servers should send files with different names (e.g., a.txt sent by nickelback, b.txt sent by metallica, c.txt sent by dragonforce). When the download is complete this process will display a message indicating so. Also, the remote machine will display a message in its user interface indicating that a file (e.g. a.txt, b.txt or c.txt) has been downloaded. Upon completion, a message is displayed. If the download fails for some reason, an error message should be displayed. When a download is occurring, a message should be displayed on the local machine. If the download fails for some reason, an error message should be displayed on the remote machine.

Similarly as what UPLOAD requires, at the end of each successful download, you will print Rx rate at which the Receiver downloaded those files and the Tx rates at which each Sender sent those files.

NOTE: Both notifications and data sending should use TCP connections. Make sure the clients send their files parallel even they cannot start sending files exactly at the same time.

10. CREATOR Display your (the students) full name and UB email address.

<*NOTE:* Undergraduate students are not required to do the following analysis. However, you will get an extra 10% bonus for this project if you complete it. >

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, 60, 70, 80, 90, 100} 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 50 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 50 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 Install iperf and Measure Network Bandwidth

Iperf is a tool for measuring network performance, especially the maximum TCP/UDP bandwidth available. For more information about the tool, refer to its homepage.

Install iperf :

1. On a CSE student sever (e.g., timberlake) download iperf source, by using this command:

    wget http://sourceforge.net/projects/iperf/files/iperf-2.0.5.tar.gz

2. Untar the source and examine its README.

3. Execute ./configure

4. make (Do not go for the make install step)

5. Execute the iperf binary (i.e., executable) present in the src subdirectory as follows:

Note: To measure the available bandwidth, you run iperf on one machine which acts as a server and another which acts as a client. The client sends packets (TCP/UDP) to the server as fast as it can, and measures the network bandwidth. For more information regarding this, refer to iperf help.

To execute iperf to measure the network bandwidth available between two machines say timerlake (server) and metallica (client), run the iperf binary as follows(you may add TCP port number which is not showed in this example):

On the server:
timberlake {~/iperf-2.0.5/src} > ./iperf -s

Then on the client:
metallica {~/iperf-2.0.5/src} > ./iperf –c timberlake.cse.buffalo.edu

Analysis: Observe the available bandwidth which iperf outputs for 1 client and then increase the number of clients from 1 through 3, by running the iperf on other client machines also as you did on metallica above.

Compare the results (available bandwidth) of iperf for a single client and for multiple clients (load variations) with the results obtained from your File Sharing Application (section 4.3). Justify why do/don't they conform to each other. Briefly record all your analysis and observations for sections 4.1, 4.2, 4.3 and 4.4 in a file called Analysis-A1.txt or Analysis-A1.pdf and turn in with your implementation.

5. Submission and Grading

5.1 What to Submit

Your submission should contain a tar file – A1.tar containing:

- All source files.
- A README describing how to compile and execute your file sharing application.
- Your analysis for sections 4.1, 4.2, 4.3 and 4.4 in a file named Analysis-A1.txt/pdf.

Please do not submit any binaries or object files.

5.2 How to submit

Use the submission command, submit_cse489 or submit_cse589, to submit the tar file.

5.3 Grading Criteria

- Correctness of output
- Your analysis
- Organization and documentation of your code