Project 1: Public *traceroute* Server

Assigned: 9/7/12, Due: 11:59pm on 9/25/12 (Tuesday))

Project Goals

The **traceroute** utility can be used to infer the end-to-end paths packets take. It is commonly used for network diagnostics and for researchers interested in creating Internet topologies. Since it is hard to create Internet-scale topologies from one physical location, public traceroute servers are often used to infer end-to-end paths from additional vantage points. In this project you will create a public traceroute server, which will help you learn sockets programming. Before writing your client program, you will connect to the server using the **telnet** utility.

Project Specification

Expected Server Functionality:

Your server should run on the CS Linux machines on a specified port. The operator of this server would invoke it as: "./*tracerouteServer* [option1, ..., optionN]". The possible options are (if a particular option is not specified, its default value should be used):

- PORT <port number>
- RATE <number requests> <number seconds>
- MAX_USERS <number of users>
- STRICT_DEST <0 or 1>

The above options may be implemented using either a simple list of key/value pairs or using Unix style arguments as show below:

- Key/value pairs: ./tracerouteServer PORT 1025 MAX_USERS 4
- Long Unix style: ./tracerouteServer --port=1025 --max_users=4
- Short Unix style: ./tracerouteServer -p 1025 -u 4

Details of each option are given below:

- **Listening port (PORT):** The operator should be able to specify what TCP port the traceroute server listens on. Possible values: 1025 - 65536. Default: 1216.

- **Rate limiting (RATE):** Public traceroute servers could be an avenue for abuse if their usage were not restricted. To guard against that, the server operator should be able to specify the number of traceroutes that can be issued by a given client in a given number of seconds. Traceroute requests issued in excess of this rate should be discarded, and the occurrence of this condition logged at the server. The client should **not** be disconnected for violating rate limiting. Instead, they should receive an error message indicating their command was not processed. Default: 4 requests per user per 60 seconds.

- **Maximum number of concurrent users (MAX_USERS):** Only a given number of users should be able to connect to the traceroute server at a given time. Above this threshold, connecting users should receive an error message indicating the server is busy before the connection is then terminated by the server. Such refused connections should be logged at the server. When a user disconnects, a slot should be opened for another connection. Default: 2 users.

- **Target IP addresses (STRICT_DEST):** Your server should have the option of allowing the operator to ensure that the users are only allowed to send traceroutes to their own IP address. This is to reduce abuse complaints or to prevent its usage in DDoS attacks. The operator should be able to toggle whether this restriction is in place. Use "0" to denote the restriction is not in place and "1" to denote it is. If a user attempts to traceroute an IP other than their own when STRICT_DEST is enabled, they should receive an error message explaining the restriction and the event should be logged at the server. The client should **not** be disconnected for violating STRICT_DEST. Default: 0.

Administrative log: The administrative log reports user activity and system behavior for the server operator. All entries should be prefixed with the time in **MM/DD/YY HH:MM:SS** format, where "MM/DD/YY" is the date and "HH:MM:SS" is the timestamp in international format (HH can take values from 0-23). Whenever a user issues a traceroute, an entry should be written indicating the destination machine's IP address or host name. When a file is tracerouted, an entry should be recorded for each command processed from the file. The log file must also contain information about server startups, connections and disconnections from users, events as described by the RATE/MAX_USERS/STRICT_DEST instructions, and any other information regarding user behavior or system events. Document all the types of messages that can appear in your log file.

Expectations on concurrency: We require that each connecting client have a seperate thread or process spawned to handle its interactions. No possible execution paths of these concurrent threads or processes should lead to an error. Specifically, ensure that you protect all functions and library calls explicitly unless they are noted to be thread safe. As an example, when updating the administrative log file, make sure that events corresponding to multiple clients are recorded correctly. All memory should be freed when your server program exits. Specificaly, there should not be any memory leaks or zombie threads or processes. Do not rely on the OS to clean up terminated threads or processes.

Expected Client and Server Interaction:

When testing and demoing your server, you should use the Linux **telnet** utility. At the end of the project, you will write a simple client that provides the basic functionality of the **telnet** utility. In order to troubleshoot, you should write the client after confirming the server works properly using **telnet**.

When clients connect to the server, the server should provide separate and concurrent sessions for each user. This can be accomplished using a new process for each user (via fork) or via threads. Each connected user should be able to run the following commands:

- **traceroute [destination machine]:** Your server program should take the destination machine as an argument. This can either be an IP address or a DNS host name. The server will then execute the traceroute utility on the supplied destination and display the result to the user. You should **NOT** develop your own traceroute implementation: reporting the results of the traceroute utility is sufficient. The server must confirm that the destination machine parameter is syntactically valid (composed only of letters, numbers, the ".", and the "-" characters), but the traceroute command output can be used to report other errors (for example, invalid host name/IP address).

- **traceroute [file name]:** A user can optionally provide a "filename" instead of a host name or IP address. Assume that this file is present in the server's local directory (the client could have previously uploaded it via alternative means like FTP). If the file exists, continue as described next. If the file does not exist, your program should assume it is a destination machine's host name and proceed according to the above bullet. The file must contain one or more commands in the format "traceroute [destination machine]", with each command on a separate line. Your server should execute the traceroute utility sequentially for each command and display the results. Separate each result using a distinguishable delimiter (for example, "----------"). This command **IS** subject to rate limiting. If the rate limit is exceeded while processing the file, all subsequent entries in the file should be discarded and logged as indicated in the rate limiting discussion.

- **traceroute me:** Your server program should perform a traceroute using the client's hostname / IP address as the target. This should otherwise be identicaly to the first option.

- **help:** Your server should display the various modes it supports if a user types "help".

- **quit:** If the user types the "quit" command, your server should close the TCP connection, disconnecting the client.

- **Invalid Commands:** If the user enters a command not listed, they should receive an error message. This message may indicate they should try the "help" command.

- **Automatic Time-out:** In addition to the above commands, the server should monitor the connection for inactivity. If more than 30 seconds elapse since the user last entered a command, valid or invalid, the server should print a message indicating the user's session has expired and then close the TCP connection, terminating the connection. Automatic time-outs should be logged at the server.

Resources and Restrictions

Begin by familiarizing yourself with the **traceroute** utility. Public **traceroute** servers, of the kind you will develop in this project are available at http://www.traceroute.org/.

You are required to use either C or C++ for this assignment. Additionally, you must use the native Linux/BSD socket system calls. You may not use other socket libraries. No credit will be given to solutions that do not adhere to these requirements. These restrictions are being made so you become familiar with the details of lower-level socket programming. Libraries other than socket libraries (dealing with options for example) will generally be acceptable; however, when in doubt ask the instructor or an AI.

As always, you are encouraged to avail yourself to Internet resources and Linux manual pages when completing the assignment. Socket tutorials such as, http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html will be helpful in understanding socket programming. However, **you MUST ensure you write your own code** and that you **CITE what resources you use**, including web tutorials and discussions with individuals outside of your group.

The following system calls will likely be required to complete the assignment: **accept**, **bind**,

close, **dup2**, **fork**, **gethostname**, **getprotobyname**, **htons**, **listen**, **recv**, **send**, **setsockopt**, **socket**, and **waitpid**. Before getting started, you should consult the manual pages for each of these system calls. To perform the traceroute, you can combine the **dup2** and the **system** system calls to invoke the traceroute utility and redirect its standard output and standard error to a TCP socket. Some other approaches could be to use **fork** and **execve** or **popen** to run the program. Other approaches are also acceptable. Concurrency can be handled with processes using fork or with threads using the pthreads library.

Build your program incrementally. Here is a road-map to use:

1. Create a "hello, world" server. This server could just accept a connection, print a message, and disconnect the user.
2. Expand the program to echo user input.
3. Allow a user to enter a destination machine and output the results of a traceroute to that IP or host name.
4. Add support for reading from a file.
5. Add policy options and command line argument reading.
6. Add logging functionality.
7. Handle concurrency requirement.

Deliverables and Grading

Submit your code, and project files as a single archive file (.tar or .tar.gz file formats only) via OnCourse. Shorly after the submission deadline, demo slots will be posted on the Demonstration Scheduling System (a reminder will be posted on the Web Board). You must schedule an appointment to demonstrate your project. Groups that fail to demonstrate their project will not receive credit for the project. If a group member fails to attend his or her scheduled demonstration time slot, it will result in a 10 point reduction in his or her grade.

In addition to testing your code for various test cases, the AIs will be explicitly evaluating the contributions of individual project partners. In cases where they determine that partners have not contributed equally, differential grading will be used. The instructor and the AIs reserve the right to determine appropriate penalty in such cases.