MPCS 54001, Winter 2021

Project 1[1]

**Due: January 21 at 5:30 pm**

Submit your assignment to your GitLab repo, in a folder named "project1". This work must be entirely your own. If you need help, post questions to Ed Discussion and/or visit the staff during office hours. As a reminder, if you make a public post on Ed Discussion, please don't give away the answer!

---

## Task

The goal of this project is to introduce you to the following skills that you will need to complete the other projects in this course:

- Navigating the departmental Linux machine environment
- Basic socket programming
- Project Testing
- Source code submission via GitLab

*Your project must be implemented in Python 3.x.*

## Echo Protocol

In this project, you will implement a basic TCP-based client and server. The client will send a line of text (i.e., a string ending with "\n") to the server over TCP and the server will *echo* that line of text back to the client over TCP as a response. The message that the client sends to the server is the *echo request*. The response that the server sends back to the client is the *echo reply*. *The client and server should be able to both run on the same machine or run on different machines.*

[1] Adopted from a Networks project written by William Conner

## Echo Server

The server shall do the following:

- Use TCP sockets
- Accept server port number as the only command-line argument
- Wait for incoming connections from the client in an infinite loop
- Print the contents of received lines of text to the standard output stream (`STDOUT`)
- Send received lines of text back to the client (i.e., echo back to the client)

Although not required for this project, the next project will require a multi-threaded server. To get an early start on familiarizing yourself with multi-threaded programming for the next project, your server *may optionally* spawn a new thread (or fork a new process) to handle each incoming echo request.

Example usage:

- `python echo_server.py <port>`

## Echo Client

The client shall do the following:

- Use TCP sockets
- Accept the server hostname as the first positional command-line argument
- Accept the server port as the second positional command-line argument
- Wait for the user to enter lines of text to standard input (`STDIN`) in an infinite loop until the user hits `Ctrl-C` or otherwise kills the echo client process
- For each line of text entered by the user
    - Open a connection to the server (i.e., TCP socket)
    - Send that line of text (ending with "\n") to the server
    - Read line of text from the server
    - Print server's response to `STDOUT`
    - Close connection to the server

Example usage:

- `python echo_client.py <host> <port>`

## Deliverables

Create a directory named `project1` in your individual GitLab repository. Upload your source code (server source files, client course files, REA DME file) to that directory. You should verify that your files were submitted correctly using the web interface at https://mit.cs.uchicago.edu.

## Grading

This project will be worth 5% of your overall grade (out of the 40% for projects) for the course since most of the code is provided in the various links under the *Resources* section of this project description.

We will run your code submissions and grade them using a combination of `netcat` and/or our course staff implementations.

General grading rubric for Project 1 is as follows (total 100 points):

- Echo server functionality: **45 points**
    - Starts/Runs: **5 points**
    - Listens for incoming connections: **5 points**
    - Receives echo request over TCP from client: **10 points**
    - Sends echo reply over TCP to client: **10 points**
    - Prints echo request from client to `STDOUT`: **5 points**
    - Handles multiple echo requests without terminating: **10 points**

- Echo client functionality: **45 points**
    - Starts/Runs: **5 points**
    - Sends echo request over TCP to server: **10 points**
    - Receives echo reply over TCP from server: **10 points**
    - Prints echo reply from server to `STDOUT`: **5 points**
    - Handles multiple input lines from user without terminating: **10 points**
    - Opens and closes connection for each request: **5 points**
- Style points, grader discretion (efficiency, code correctness, documentation in README, comments, or other considerations): **10 points**

## Testing

You can perform local testing on the same machine by having your echo client talk to the echo server with your local IP address as the host command-line argument for the client.

You can simulate an echo server with `netcat` (for testing your echo client) by running the following command[2].

```
$ mkfifo fifo
$ cat fifo | nc -k -l <port> -v | cat > fifo
```

You can simulate a single iteration of an echo client with `netcat` (for testing your echo server) by running the following command. Note that your echo client implementation must allow the user to enter multiple lines of text over multiple iterations.

```
$ nc -v <host> <port>
```

[2] Taken from https://unix.stackexchange.com/questions/147977/netcat-echo-server-possible-with-pipes-instead-of-commands-as-strings

## Hints

- The well-known port range (0 through 1023) cannot be used for your echo server, so you will have to select a port number between 1024 and 65535.
- You all share the same CS department machines (`linux1`, `linux2`, and `linux3`). If you start your echo server and get an error about a port being in use, just try another port number greater than 1023.

## Resources

As mentioned during the sockets recording, the following resources are available to learn socket programming. Most of the assignment can be completed by adapting sample code from those links to meet the specifications of this project.

- https://docs.python.org/3/howto/sockets.html
- https://docs.python.org/3/library/threading.html

To access a Linux machine, you can ssh to `linux.cs.uchicago.edu` and it will send you to a suitable host. Those machines support running Python 3 files.

I strongly suggest developing on the Linux machines, rather than using your own personal machines, since that is where your projects will be graded. Any differences between that environment and your personal computers are ultimately your responsibility.