CSC45500 Programming Project #4
Due: Friday, April 28, 2023, 11:59PM

Objectives
- build a network socket based server in a Linux system
- build a network socket based client in a Linux system

Problem Statement
You are to write two programs: a server program and a client program. The server program will receive commands to return, clear, or add a value to an integer accumulator within the server. The client program should send commands to the server for the server to process

The Server Program
The server program should:
- take a single command line argument representing the integer port number for the server to utilize on the local machine.
- initialize its integer accumulator to 0
- use the concurrent server model using pthreads to process TCP connections.
- repeatedly process an incoming ('\n' terminated) character string from a connected client
- if the first word in the string is:
  - **get**, then the server should return a string containing the current integer value of the accumulator to the connected client.
  - **add**, the the server should read the next string found in the character stream sent by the client. This second string will represent an integer number that should be added to the accumulator. The server should then return a string containing (only) the new value of the accumulator in a message to the connected client.
  - **clear**, then the server should simply set the accumulator to 0. No message should be returned to the connected client.

The Client Program
The client program should:
- take two command line arguments. The first argument should be the IP address (or name) of a server with which to connect. The second is an integer representing the port number.
- The program should then prompt the usr to enter a *single* command to send to the server, namely one of:
  - **get**, which should send the associated command to the server and print out the value of the accumulator that the server returns.
  - **clear**, which should cause the server to reset its accumulator value to 0. The client should not print anything additional out as a result of running this command.
  - **add <intval>**, where <intval> would be an actual integer value. The integer value should be added to the server's accumulator, and the client should print the new value of the server's accumulator.

Each of the commands should be terminated with a '\n' character.

Example Execution
Suppose you invoked the server on machine with IP address alderaan.lindenwood.edu (this is one of the machines in the lab in front of my office) as follows:

./proj3server 50000

Note that the above will effectively enter an infinite loop, waiting for incoming clients. Next, suppose the client is run on the same machine as follows as follows:

./proj3client 127.0.0.1 50000
get
0

In the above example, the user specified the “get” command and the server on the local machine (127.0.0.1) responded with “0”, which the client printed out.

Suppose that next, another machine (on campus) ran the following:

./proj3client alderaan.lindenwood.edu 50000
add 42
42

In the above example, the server was told to add 42 to its accumulator and return a string representing the new value of the accumulator. The client then prints the string that was returned.

Then suppose the following is run:

./proj3client alderann.lindenwod.edu 50000
add -13
29

In the above example, the server was told to add -13 to its accumulator and return a string representing the new value of the accumulator. The client then prints the string that was returned.

Then suppose the following is run:

./proj3client alderaan.lindenwood.edu 50000
get
29

In the above example, the server was told to a return a string representing the value of the accumulator. The client then prints the string that was returned.

Then suppose the following is run:

./proj3client alderaan.lindenwood.edu 50000
clear

Note that nothing is returned from the server, so nothing is printed.

Then suppose the following is run:

./proj3client alderaan.lindenwood.edu 50000
get
0

In the above example, the server was told to a return a string representing the value of the accumulator. The client then prints the string that was returned.

Note that these tests are nowhere near exhaustive. *It is 100% up to you to come up with more exhaustive tests than the above!*

What To Hand In

You will be submitting a zip or tgz file containing your source code (.cpp and .h files) and a `read.me` file to Canvas. Make sure that you place the project into a single folder (which may contain sub-folders).

The `read.me` file should include information about your project including (but not limited to):
- your name
- the date
- the platform you developed your code on (Windows, Linux, …)
- any special steps needed to compile your project
- any bugs your program has
- a brief summary of how you approached the problem

You might also want to consider adding things like a "software engineering log" or anything else you utilized while completing the project.

Grading Breakdown

Correct Submission	10%
Code Compiles	20%
Following Directions	20%
Correct Execution	40%
Code Formatting/Comments/read.me	10%
Early Submission Bonus	5%

Final Notes & Warnings

- This is not the kind of project you can start the night it is due and complete on time. My recommendation is to start *now.*
- *Start now!*
- Projects may *not* be worked on in groups or be copied (either in whole or in part) from *anyone* or *ANYWHERE*. Failure to abide by this policy *WILL* result in disciplinary action(s). See the course syllabus for details.
- *Have you started working on this project yet? If not, then* ***START NOW!!!!***