Sockets Programming Assignment 1

TCP and UDP Applications Programming - Sensor Networks

For this assignment you will design and write your own application programs using network sockets. You will implement two different versions of the application. One version will be based on TCP. The other will use UDP. You will implement client and server applications for both versions (4 applications total). You may choose Java, Python, or C/C++ for your implementation. You will be using Docker, a container technology for developing, shipping, and running applications. The container image we are providing will support development using all of these languages, and also has support for packet captures using tshark (command line version of wireshark).

The application you are designing is a simple sensor application that allows you to collect measurements from sensors and receive historical data about those measurements. The historical data will include the minimum, maximum, and average sensor values sent by each sensor, as well as an average for all sensors.

You will include a basic online authentication service to protect the quality of the data you are collecting and to protect unwanted viewing of the sensor data by others. The authentication service is based on a shared secret “password” and a challenge response protocol. Each sensor will have an identifier (username) and a unique password for that identifier. These will be used to determine whether or not the sensor data is accepted and whether any data should be sent from the server to the client.

The client command should be called “sensor” and it should read the username, password, and sensor value from the command line. If the authentication is successful, the command should report the sensor value received by the server and the historical data calculated and returned from the server. If the authentication is not successful, appropriate diagnostic feedback should be sent to the client and recorded by the server. For instance, if you provide an invalid password it should print “User authorization failed.” as output.

You must allow the host name (or IP address) and port number of the sensor server to be specified on the command line of the client in addition to the username and password. (Of course, you would never implement a real security system that specifies username and password on the command line. In such cases another user running the “ps” command would quickly learn your password!)

For instance, the command:

```
# sensor-tcp -s 172.17.0.3 -p 8591 -u “Room100SE” -c “eye<3sockets!” -r 68.2
```

might produce the output:

```
Sensor: Room100SE recorded: 68.2 time: Sep 13 00:47:48 sensorMin: 67.4 sensorAvg: 68.0 sensorMax: 6
```

This transaction should be sent from the sensor client application to the sensor server application. Each application should be running in a separate Docker container. In the example above, the server container has an IP address of 172.17.0.3 and is listening on port 8591. Note: The command line must work exactly this way to make sure that we can easily test your program.

When you start the server, you should specify the port number on which it will be listening as well as a file containing the sensor identifiers and passwords. Your server should support at least three different sensors. The passwords file should be a csv file of the form: sensorID,sensorPassword (no spaces should be allowed in either string.)

```
# sensor-server-tcp -p 8591 -f passwords.csv
```

This would start the server on port 8591. The server should run forever, processing multiple sensor client transactions in succession. The server should return the same information as the client when a sensor authenticates and sends data successfully, and an appropriate error message followed by information about the client when the authentication fails. (ex: “User authorization failed for client: 172.0.X.Y port: Z user: upsidedown”).

Challenge Response Algorithm

The access algorithm you are implementing is an extremely simplistic form of the Digest Authentication scheme widely used by web servers and many other services including VoIP. You should start by reading about these in Chapter 8 of your text and also in the Wikipedia pages on Digest Authentication and MD5 Hash Function.

Your access application will require the exchange of four messages.
- Client sends “authentication request” message.
- Server responds with a one time use, challenge value in the form of a random 64 character string. (You get to decide how this random string is generated.)
- Client computes a MD5 hash of the string formed by concatenating the username, password and the random string sent by the server. Hash = MD5(“username”,“password”,“challenge”)
- Client sends the clear text “username” and the resulting “Hash” to the server.
- The server takes the username, finds the corresponding password on file, and performs the same MD5 calculation. It then compares the calculated Hash to the one sent by the client.
- If they match, the user has successfully authenticated. If no match, then authentication fails.

You will need to develop your own “protocol” for the communication between the client and the server. While you should use TCP or UDP to transfer messages back and forth, you must determine exactly what messages will be sent, what they mean and when they will be sent (syntax, semantics and timing). Be sure to document your protocol completely in the program writeup.

Your server application should listen for requests from sensor clients, process the data sent, and return the results (min, avg, max). After handling a client the server should then listen for more requests. The server application should be designed so that it will never exit as a result of a client user action. You should also be prepared for multiple requests arriving from different clients at the same time.

Your server will “store” the password along with the username and the data required to compute the results in memory. Keep it simple, you do not need to use a database. When you restart the server, the results should not be remembered from a previous run (which is not a realistic scenario) but will help us with grading.

I’m asking you to implement the MD5 hash. However, you don’t have to write the code. There are numerous versions already implemented that you should use. For instance, the RFC has one. You can also find example code here. However you choose to implement it, you must include a citation in your source code indicating where your MD5 came from. Note that you don’t have to fully understand the MD5 algorithms to be able to use this in your program! You just need a function that will calculate the hash from your string.

Focus on limited functionality that is fully implemented. Practice defensive programming as you parse the data arriving from the other end. Again, don’t focus on a powerful database or a fancy GUI, focus on the protocol and data exchange and make sure that you deal gracefully with whatever you or the TAs might throw at it.

Grading

The grading will be divided as follows:

20% Documentation
40% TCP Implementation
40% UDP Implementation

For both implementations the grading will be broken down as follows:

10% - authentication handling
10% - sensor transaction and results
5% - multiple transactions and results from same sensor
5% - multiple transactions and results from different sensors
10% - resilience/defensive programming

Extra credit is available for:

5% - handling multiple transactions from different sensors sending simultaneously
10% - Server support for UDP and TCP clients simultaneously

Tips

You are implementing two complete, separate versions of this assignment: a client and server using TCP and another client and server using UDP. Your final submission should include 4 separate executables. They should be called: `sensor-tcp`, `sensor-server-tcp`, `sensor-udp` and `sensor-server-udp`.

Your TCP and UDP versions will share much of the same code for command line parsing, password storage and user interaction. However, your UDP implementation will have to deal with lost request messages. Make sure you consider what happens when a message is lost. You will need to handle this in some way such as using a timer to retransmit your request. A good way to test your client in this situation is to run it without the server running. Does your client handle this gracefully?
You should include a “-d” command line option that enables debugging messages to be printed. Such as the client printing “Sending authentication request to server ”, “Retransmitting request after ”, “Sending username and hash to server”, “Sending random string to client” etc. Without the –d, you applications should only output as specified in the description above.

A portion of your grade will come from the TA running your program in a Docker container using the provided image. You must clearly document how to run your program in your README (and Makefile if necessary).

Make sure that you do sufficient error handling such that a user can’t crash your server. For instance, what will you do if a user provides invalid input?
You must test your program and convince me it works! Provide me with a sample output (you can use script or cut-and-paste) that shows all of your functions working. If you fail to demonstrate a capability of the program I will assume that feature does not work.

Please save your work as you go. This includes making copies of things that work so that you have something should your future work cause previous work to fail. Tools like GIT are useful for this, but even a periodic zip of your directory is better than nothing.

Notes for deployment

First you will need to set up Docker on your machine. Start by installing Docker using the instructions below for your platform:

For Mac: https://docs.docker.com/docker-for-mac/
For Windows: https://docs.docker.com/docker-for-windows/
For Linux (Ubuntu preferred): https://docs.docker.com/engine/getstarted/
All other information: docs.docker.com

Once Docker is installed, follow the instructions below to pull the container image for the project:

Pull the created Docker image:

```
$ docker pull gt3251/project1
```

Run the following command to check if the image was pulled from the Docker repository

```
$ docker images
```

Run the image after confirming that it was present

```
$ docker run -t -i gt3251/project1 /bin/bash
```

Now you are in the container at a shell prompt. You can check if Java, python and gcc are installed in it by running the following commands:

```
$ which javac
(You should see /usr/bin/javac in shell)
$ which python
$ which gcc
(Same goes for the other two)
```

To exit from the container, run:

```
$ exit
```

To check the container id, run:

```
$ docker ps -a
```

Note the container id. Every time you exit, you can stop the container by running:

```
$ docker stop <container id>
```

You can restart the same container by running:

```
$ docker start -i <container id>
```

Go to the Docker Settings -> Preferences -> File Sharing -> Create a folder `CS3251_PA1_<lastname>_<firstname>` on your local machine (PC or laptop you use) and add it in the file sharing options. Restart Docker.

Now start two containers, one for your sensor client and another for the server. In one window Run:

```
$ docker run -t -i -v /<Path to your project directory>CS3251_PA1_<lastname>_<firstname>:/home/CS325
```

And in another window run:

```
docker run -t -i -v /<Path to your project directory>CS3251_PA1_<lastname>_<firstname>:/home/CS3251_
```

You can run `ifconfig` in each window to find the IP addresses assigned to each container.

You will find your files in `/home/CS3251_Proj1/myfiles` in both containers. You can edit, compile, and run your programs from within this folder in either container as well your host operating system. We recommend this approach so that your changes are not left inside the container, and so that you don't have different versions of files in different containers.
When you exit the container you can either restart it with “docker attach” or start a new container as you started these. You can look into the “docker ps” to view the container identifiers and names. Note that any changes you make to the environment will only be retained with attach and not by creating a new container.

You are free to write your code using any IDE or text editor on your local machine. You can even run and debug your applications on your host machine, but you are responsible for making sure your applications run on the provided image in a separate Docker containers for clients and servers for grading purposes. We highly recommend that you don't wait until the last minute to run your code in the Docker container environment, and suggest periodically running in a container even if you are debugging elsewhere.

Program Submission Instructions

Turn in a zipped file - `CS3251_PA1_<lastname}>_<firstname>.zip` (ex: CS3251_PA1_Burdell_George.zip), containing your well-documented source code, a README file and a sample output file calledSample.txt. Only .zip is allowed.

The README file must contain:
* Your Name and email address
* Class name, section, date and assignment title
* Names and descriptions of all files submitted
* Detailed instructions for compiling and running your client and server programs including a Makefile if necessary.
* A description of your application protocol (1/2 to 1 page) with sufficient detail such that somebody else could implement your protocol
* Any known bugs, limitations of your design or program

You will be graded on the correctness of the code and its readability and structure. As you write the documentation, imagine that I intend to implement your protocol in my own client that will interoperate with your server. Make sure I don't have to read your source code to figure out how to do this.

We will unzip your file and mount it when launching Docker containers consistent with the instructions above. Make sure your code runs on Docker.

By 11:55 PM on the due date, you must submit your program files online:

- Create a ZIP archive of your entire submission named `CS3251_PA1_<lastname}>_<firstname>.zip`
- Use T-Square to submit your complete submission package as an attachment.