Devante Wilson
100554361

Lab #3: Distributed Systems with JeroMQ

SOFE 4790U

Task #1: Establish the connection

1. To accomplish this task, the code in both the client and the server where modified to display a 'connection successful' message upon client connection to the server. A string containing a message was created on the client then converted to a byte stream and sent through the socket on the client. On the server, it blocked until the message was received – the message was converted back to a string for displaying. Meanwhile, the client is blocked while waiting for a reply; the server sends back a message the same way as before – the client displays this.

2. A challenge I faced during this task was not realizing how to handle blocking of either the client or server when waiting for a message. I solved this by looking through the Java documentation and JeroMQ methods. I found some useful methods, *Thread.currentThread().isInterrupted()* to detect a whether the receiver has a pending interrupt request. For JeroMQ, *ZMQ.Socket.recv() and ZMQ.Socket.recvStr()* are equally as useful to receive the actual message, into a byte array or a string, respectively.

3. This task helped me learn how to correctly work with JeroMQ sockets to communicate between client and server.

Task #2: Prime number

1. In this task, the code was modified from the previous task; an integer was sent from the client and parsed properly in order to read the value on the server. A method was created on the server to compute the prime numbers up to the provided number. The numbers were sent back the client for displaying.

2. A small obstacle I encountered was parsing the numbers back and forth between integer, string, and byte streams. I quickly referred to the lab instructions where I learned that ZeroMQ is unaware of the data sent except for its size in bytes. Thereafter, I was able to format the messages safely.

3. I learned how to safely parse message content into correct format for retrieval.

Task #3: Repeated Inputs from Client

1. This task required continuous messaging to occur between the client and server. Thus, the message passing portion and receiving keyboard input was performed inside of a

while loop – the condition checked for valid input. On both the client and the server, an identical method was formulated to perform the string reversal. The methods were called upon receiving a message on their respective end. Since the user should be able to type 'close' to the server to stop the process and terminate the connection, an 'if' statement was made to handle this event. In retrospect, the loop could have been made to handle this condition instead.

2. This task did not present much of an obstacle. I was already aware how to handle continuous requests from the client based on previous assignments and lab tasks.

3. I learned how to stop a process and terminate the connection in JeroMQ.

Task #4: Subscribe-Publish

1. This task required the implementation of a publish-subscribe design pattern using JeroMQ. A set of clients subscribed to a server by sending a different zip code as keyboard input. The server continuously broadcasted the update of the population in the area given by the supplied zip code. To complete this, the message passing occurred inside a loop on the server. A method was created to randomly generate a population size once a generated zip code matched the client-supplied zip code. The population was sent back to the client for printing.

2. This was my first attempt at implementing a publish-subscribe service. Thus, I needed to ask the TA for clarification on how the architecture should function.

3. I learned how to create a simple publish-subscribe service using JeroMQ.