Sidharth Gilela
sgilela@ucsc.edu
Assignment 3
CMPE 156

Design Document

Goal:

The objective of this assignment is to implement a client server model such that the client(s) are all multi-threaded and the server is concurrent. Additionally we had to implement this over a UDP network. This means that we had to implement reliability over the UDP network. This was a great learning experience in that I got to experience the details of UDP. Now, I know a majority of the advantages and disadvantages it provides. I additionally know the complexity of TCP to ensure reliability during packet traversal. All in all, I had a great time working on this assignment and it was a great learning experience.

Compile:

To compile the server and client run these commands from the src folder:

```
make udp-send
make udp-rcv
```

To compile both the client and the server together run:

```
make all
```

Design:

The design of my protocol will be described through an example run between a couple clients and servers running at the same time. The first thing each client does is send a message to each of the servers on the server list file. The servers that respond back with a message are the ones currently active. The servers may respond with an error message when they cannot open the file requested by the client. When the server receives this first message, it does a fork and creates a new port number inside the child and a new socket. The parent's socket is closed in the child. This new port number that is created in the child will be inputted into the buffer being sent back to the client. Other information that gets sent back include the file's size, ACK number and more. The client then stores this port number and the current ip address being tested with in a separate array. These values will be referred to later. As a heads up, during any communication between the client and the server messages are sent with a title meaning they are sent as "WantConnection filename:myFile seqNum:mySeqNum". Appropriate messages are sent for appropriate communications between the client and the server. Another note is that during any communication with the server a sequence number is sent to the server. The server will then always send back an ACK number along with other info requested to the client which is equal to the sequence number that it parses on the server side. The client when receiving this ACK number will check it with its current sequence number. If the numbers match no retransmission is done and the sequence number on the client side is incremented. If the numbers do not match, then a retransmission is done. This means that the client will send out that same message to the server with the same sequence number as well. This time though the timeout value for the readFrom system call is increased. Due to the idea of exponential back off, the client will keep increasing its timeout value by exponents of 2 meaning 2^1, 2^2, etc. The client will retransmit at a maximum of 3 times waiting upto 8 seconds for a response on the final retransmission. Keep in mind during any communication between the client and server, this process is done to ensure reliability and thus, I will not constantly mention this as I continue to describe my protocol. Now, the client keeps the port number in an array of port numbers as I was saying. Each thread when created is given one of these port numbers and the corresponding ip address. After figuring out the minimum number of connections, it will now send information to the server using pthreads and the server will respond after making the appropriate calculations. What I mean by this is that the client will have a while loop, constantly asking for information from the server until it receives all the data that client thread is expected to receive. During each of these messages a sequence number and ACK are sent and received. This is to ensure reliability and retransmissions if need be. A reason why I chose the three transmission times to be 2 seconds, 4 seconds, and 8 seconds is because after searching online, what seems to be an approximate server response time is 1 second. I have increased the max timeout to 8 seconds just in case there is some really ugly traffic going on and packets keep getting lost. As a side note, when a retransmission occurs I have presented a message on the screen so that the client and user knows that a retransmission is happening. I have additionally made the server robust by not exiting when the client exits at some random moment. After the client receives the last message, it will know to exit because the recvFrom will be a 0. When this occurs, the client exits. Another error handling case is that if the server-list is empty or no servers are able to respond, the client exits and presents a message on the screen. If an output file has been created, the output file is deleted and reconstructed during the second time the user wants to run the code with the same file. Thus the output file does not need to be deleted during every run.

Note:

This assignment is not considered to be late because I have spoken to the professor about submitting this assignment a day late and he has told me there will be no penalty.