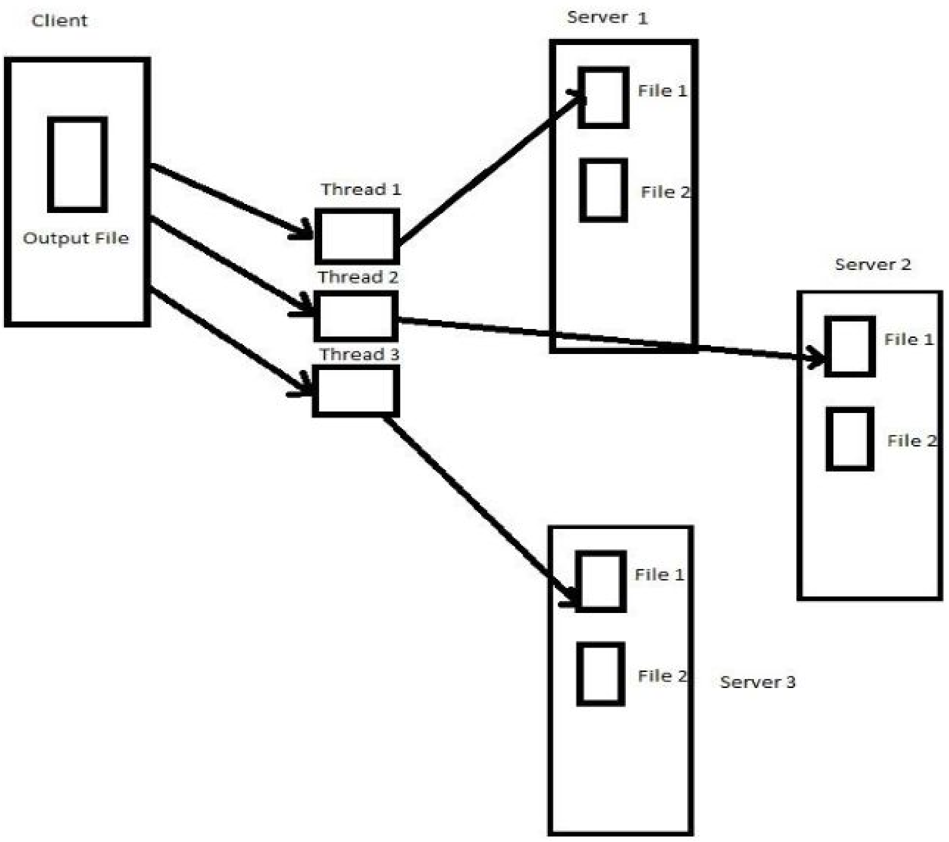
Naishil Shah
nanshah@ucsc.edu

CMPE 156 - Homework Assignment 3

Theory:

The goal of the assignment was to use Pthreads to achieve a client server architecture where client threads access multiple server and obtain different chunks of the same file and output a final combined file at its output. This transfer of data has to be done concurrently and the client threads should write the incoming data from their respective servers on the same output file. Shown below is a diagrammatic representation of the same.
This assignment is built on the second assignment, with the only difference being the use of UDP instead of TCP protocol. This gave rise to the problem of unreliability which is tackled with the use of timer variables.



Workflow of the code:

1. The servers start with their respective port numbers as mentioned in the server-info.txt file.
2. The client gets the server-info.txt file having the IP address and the port number details of the servers and the number of servers to be used by the user via command line.
3. The client parses through the text file and stores the information in arrays.
4. The client then requests one of the server (1 st one by default) for the size of a file. It waits for the server's acknowledgement/response for a particular amount of time as desired.
5. The server receives the name of the file and returns its size if it exists, otherwise it returns a string saying No such file exists. Again the server waits for an acknowledgement from the client side.
6. On obtaining the size the client then spawns threads which get their own chunk sizes and the offsets. The number of threads are equal to the the number of servers mentioned by the user or the max number of servers available.
7. The threads access different servers concurrently and retrieve the chunk they are allotted from the file on the server. Throughout the process, both the client and the server wait for each other's acknowledgement so that reliable data transfer is taken care of.
8. This data is written to the output file. This is also done concurrently.
9. Finally the client waits for the threads to finish and once they finish the client program closes.

Design:

1. The client has a mechanism of notifying the server whether it is asking for File Size or for data.
2. By sending the "CHECK" token the client tells the server that it is currently asking whether the file exists or not and if it does then what the size of the file is.
3. If the client sends the token "DATA" then the client is requesting the server for bytes of data from the file.
4. So the first message sent from the client to the server always have a HEADER with a token containing either "CHECK" or "DATA". The "CHECK" token is usually sent by the main client thread (called thread 0) and the "DATA" token is usually sent by the threads.
5. In the program there are constant messages exchanged between the client and the servers. Most of the messages sent using the *sendto* and *recvfrom* calls are followed by an ACK. This was done to ensure the client or the server does not write asynchronously and data is not out of place due to the use of UDP protocol.
6. The major difference in the application was the use of *sendto(), recvfrom()* and *select()* function calls. The *sendto()* was used in place of *write()* function call in TCP and *recvfrom()* in place of *read().* The *select()* is used to keep track of whether any

input is still available from the particular socket descriptor. This application is in accordance with the assignment specifications but does not eliminate the errors of the second assignment.

Error Checking:

The following cases of errors are checked in the application:

1. The IP addresses and the port number not specified for client.
2. The Port number not being specified for the server.
3. Error of creation of a socket.
4. Error in Binding, Accepting and Listening for the the socket.
5. The client application crashing.
6. Error in reading the data.
7. Error in opening the file.
8. Error in reading from the file pointer.
9. Checking for a client disconnection.
10. Error in converting the IP address provided as a String into a Network order address on the client.
11. Error in establishing a connection between client and server.
12. Error in writing to the send buffer in client.
13. Error in creating a Thread.
14. Error in seeking a particular a place in a file using fseek().
15. Error when Threads join to the main thread.
16. Error If the number of servers are not specified on the client.
17. Error if the number of servers specified are greater than the number of servers available.

How to Run the code :

Please refer README.txt