Joe Cowen-Richards
Anthony Fammartino
James Rodriguez
CSE434, Computer Networks, T,Th, 1:30pm

Design and Analysis

Request.h

Request.h contains the data structures to be used by both the client and the server. The client sends a request to the server. The request data structure was given to us so that all the groups used the same formatting when making requests to a server. The server saves each client into the client table. A client table is simply a vector of clients. A client contains the process_id of the client, the incarnation number, the client number, the request number, and finally a character array of the message to send back to the client. This message changes when the client sends a new character to append to the front of the character array.

Client Model: Client.cpp

The Client sets up the socket descriptor and then sends 20 requests to the Server. There is a GetIncarnationNumber function that handles the locking and unlocking of the incarnation file (inc.txt) when a request is not received. After this, a check is done; if the request is less than the randomly chosen number, there is no chance of failure. The client receives a message back from the Server and prints out the string that the Server made from the character that the Client sent to it. The Client sends the entire request data structure that was required in the spec of the project. The locking of the inc.txt file is done using the fcntl function discussed in class.

Failure rates for the client are simulated by first selecting a random number from the 20 possible requests to begin applying the 50% failure rate. Once the request number is reached for that request and every request afterwards a random number is selected between 1 and 100 and if the number is less than 50 then the failure is applied and the client does not send the request and the incarnation number is incremented.

Server Model: Server.cpp

The server saves all of the clients into a vector of client entries. This vector is sorted after every client request in order to use the find_id structure that was written to lookup existing clients. The find_id is used in conjunction with the find() function in the vector library to lookup a client based off of the client's number. If this number is found, then a pointer (named iter) is returned pointing to that client, if not, the new client is added to the back of the client table using the push_back() function in the vector

library. Once this new client has been added, a pointer is given pointing to that last element so it can be manipulated and sent back to the client after the sendMsg character array has been modified. DieWithError is a function that was given to us to handle error codes that may be thrown by the Server at any given time. CompareByClient is a function that is used to sort the client table. Since the client table is a custom structure, a simple sort() function does not work, and a custom comparison function must be written in juxtaposition with the sort() function. CompareByClient fulfills that need and keeps the vector sorted by client number.

Failure rates for the server are simulated in two places. The first: a random number between 1 and 100 is generated and compared to the value 10. If the random number generated is between 1 and 10 then a failure is simulated such that a request is completely dropped. If the random number is between 11 and 100, the request is processed normally. The second: A second random number is generated and the same probability rules applied. If that number is between 1 and 10, then the request is processed and the sendMsg attribute manipulated, but the response is never sent back to the client.