PROJECT 2
Assigned Thursday, September 21, 2006
Due Date: October 02, 2006 (Monday) 4:59 pm EST (on campus)
October 05, 2006 (Thursday) 4:59 pm EST (EDGE)

SPECIAL NOTE: THIS PROJECT IS A OPEN SPECIFICATION PROJECT. IMPLEMENTATION DETAILS AND MESSAGE FORMAT AND PROTOCOLS HAVE BEEN LEFT OUT FOR YOU TO DECIDE YOURSELF.

General Information:

This project is an individual project and NOT a team project. In this part you will extend project 1 to incorporate more advanced features such as creation of chat rooms and further learn about IP multicast and its features. From this point on the 'command module' and the 'server module' that you developed as part of project 1 will be referred as 'end user' and the 'agent' / 'user agent' module. In addition to adding more features and command to 'end user' and 'agent' you will also create a centralized communication management server which would be referred as 'server' from here onwards. Below is the basic message flow diagram explaining the above idea.

There could be multiple instances of such 'server' running independently of each other on the network. In project 1 we did not enforce name collision prevention. In this part name collision enforcement will be handled by the centralized 'server'. Server maintains information about all users and also on chat rooms and the corresponding multicast details.

The project 2 will consist of 3 parts. Part 1 will implement the 'server'. Part 2 will deal with adding more commands for the 'end user' interface. Part 3 will add more capabilities and will define more actions for the 'agent' or 'user agent' module.

Part 1: Server Module

The server module will have 2 threads. A receiver thread which will start executing as soon as the server application is started and will continue execution till the server module is terminate. The second thread will be the sender thread that will be called to send messages as and when needed and will terminate after sending the message to the intended address. The server creates and manages multicast groups. The following commands are supported by the server.

- createroom <one word name> – creates a chat room and names that chat room
- joinroom <chat room name> - allows particular user to join a chat room
- leaveroom <chat room name> - allows the user to leave a chat room
- ulist <chat room name> - lists all the users that are in a chat room
- listrooms – lists all existing chat room names
Message packet corresponding for each command possibly could have the following structure (remember these commands are coming from the user-agent to the server and the format specified is just an example):

command [argument if any] [user alias name] [ip address] [port number] [other parameters]

For example sake assume user 'a's talk receiver ip address is x.y.z.w and port number is p. Further assume its self-alias is 'a' and it wants to create a chat room titled 'geometry' then the data portion of the UDP packet created corresponding to the command 'createroom' to be sent to the server from the user agent could be

createroom geometry a x.y.z.w p

In this project we will not assume peer to peer or end to end communication between agents / users. All chat messages will be originating from end users and terminating to some chat room and not to any particular 'user' but control messages may terminate to the server. Also any user at any given time could participate in only 1 chat room. All these requirements will be enforced by the server and also by the user agent.

Server operations explained:

When the server program executes, it must ask for a startup key. This startup key should be used to generate unique seed for each chat rooms.

Whenever the server receives a command to create a new chat room. It must first check for any existing rooms with the same name. Also it must check if the requesting user is not participating in any other chat room by comparing the user agent's receiver thread ip address and port number. If no such room is found and the user is not part of any other chat room then it creates a new room and also generates a unique seed for that room. One way to do it could be by using this formula:

new chat room seed = DES(md5(current system time stamp), key)

Next it picks up a multicast address from the range 230.0.0.1 to 230.0.0.255 that is not assigned to already existing chat rooms. It then enters this tuple value <chatroom name, seed, mcast address> in its chat room database / table. Also it maintains a list of users participating in each chat rooms. If the chat room does not exist it creates that room and immediately adds that user in the user list for that chat room. It then sends back to the user a reply message specifying the chat room seed, multicast address for that chat room. If the chat room creation fails because it already exists, server returns failure message back to the user receiver thread address.

Remember – all communication from the server to the user agent is unicast UDP messages and not multicast.

When the server receives a joinroom command, it first checks if that user is already participating in any existing chat room or not. This check is done by comparing the user agent's talk receiver IP address and port number in the list of users for any existing chatroom. If the user is already participating in any existing chat room, server responds back by a failure message specifying the name of the chat room the user is already participating in. Also if the alias is already used by some other user agent in that chat room user list, the server responds back indicating name collision to the end user agent further advising it to change its self alias and try again. Otherwise it returns back a message specifying the chat room seed and the corresponding multicast address. Server also then adds this user to the list of users for that chat room.

When the server receives a leaveroom command, it first checks if that user is already participating in that particular chat room or not by comparing the ip address, port number fields of the existing users for that chat room. If the user is currently participating in that chat room then the server removes its entry from the list of user for that chat room. It immediately changes that chat room unique seed, again immediately communicates to the remaining user of that chat room the new chat room seed (using UDP unicast messages) in turns. It then indicates to the original user that it has been successfully removed from that chat room. Otherwise it returns back
a failure message. If the server changes the unique seed for that chat room it also updates the chat room table with the new seed value. Further if after removing the requesting user from the chat room successfully, if the chat room becomes empty, the server must destroy that chat room session by removing that room details from the existing chat room table (this is different from the user list table for each chat room).

When the server receives a ulist command from the user agent, it sends back the list of all users specifying their alias name along with the IP address and the port number for that chat room regardless of the requesting user participation status in that chatroom. If no such chat room exists, server returns back an error message.

When the server receives a 'listrooms' command, it responds back with the list of all existing chat rooms names to the requesting user-agent talk receiver address. All the responses sent back to the agent by the server is done by spawning the sender thread.

**Part 2: End User**

End User is synonymous to the user command module of project 1. For this project additional valid commands will be:

- `createroom <one word name>` – creates a chat room and names that chat room
- `joinroom <chat room name>` - allows particular user to join a chat room
- `leaveroom <chat room name>` - allows the user to leave a chat room
- `ulist <chat room name>` - lists all the users that are in a chat room
- `listrooms` – lists all existing chat room names
- `csend <message>` - sends message to current chat room multicast address

All other commands usage except csend has been explained in Part 1 of the project. The csend command is meant to send message to the chat group. Remember, the user agent is not performing any management operation, it is just sending these commands to the 'user agent' aka command server thread of project 1.

**Part 3: Agent / User Agent**

Agent is synonymous to the server module of project 1. Again it already consists of three threads namely command server thread which receives all commands from the end user, talk receiver thread that receives all messages from the communication manager server aka “server” described in Part 1, unicast message sender thread that sends command packets to the communication server aka “server”. In addition it also has a fourth thread namely mcast sender thread that is called upon to send multicast messages to the chat group. Also there exists a fifth thread which listens to any incoming multicast messages called mcast receiver thread. Remember: threads 3 and 4 are activated as and when necessary. Fifth thread is also started when the agent joins or creates a chat room and it terminates when the agent receives a command to leave that chat room, let us call it mcast receiver thread. First two threads run continuously when user agent is executed and terminates when it receives 'exit' command from the end user just like as in project 1.

**Agent / User Agent operations explained:**

There is no need to maintain alias table for this project because no peer to peer messaging is allowed right now. It can be deactivated. Also at any point the user agent could be participating in at most 1 chat room and no more so there is no need to maintain any chat room table structure as well.

When the agent receives a createroom, joinroom, leaveroom, ulist, listrooms command at its command server thread, it spawns a unicast sender thread passing it the command and the corresponding parameters.
The unicast sender thread then appropriately creates a UDP message packet on the lines of project 1 containing necessary information for the “server” which includes the self-alias information as well as unicast receiver thread's ip address and port number and sends it to the “server”'s receiver thread ip address and the port number.

If the agent's command server thread receives the csend command, it spawns a mcast sender thread passing to it the message as the argument.

The mcast sender thread creates a new multicast message packet to send to the chat room specified by the multicast address global variable. The constructed packet's data portion must contain the unique chat room seed followed by the message in that order. It then sends that packet on the multicast channel and terminates after successful operation, otherwise it should output an error message indicating send failure. Note there is now no need to append aliasname, ip address and port information in these packets as was done in project 1.

When the agent's receiving thread receives a message from the server, depending on the message type it takes appropriate action (NOTE: Regardless of whether an action is taken or not, appropriate output must be displayed on the console indicating the receipt of the new incoming message).

- If the received message is in response to the create room or join room command, if it is a error message it just displays the message, otherwise it updates the global chat room name variable, chat room unique seed variable and multicast address variable to reflect the most recently joined chat room details. Also it spawns a new mcast receiver thread enabling the agent to start receiving messages on that multicast channel from other users in that chat room.

- If it receives a message in response to the leave room command, if the message is an error message it just displays the message otherwise it terminates the mcast receiver thread.

- If it receives a message in response to ulist or list rooms commands, it just displays the message, possibly after parsing it to separate out different fields / components.

- If the message received is due to change of the unique chat room seed by the server then it immediately changes the global unique chat room seed to reflect the new seed.

Note: It is up to you to decide the incoming message format along with identification fields (if needed) in order to differentiate messages coming in response to one command from those in response to different commands.

When the mcast receiver thread receives a packet on the multicast channel, it first checks for the validity of the message. If the first field of the data portion matches with the current unique chat room seed, the message component is displayed on the console otherwise the packet is discarded without further action.

**SPECIAL INSTRUCTIONS:**

1. All outputs on the console must be in some easily understandable format.
2. When the exit command comes from the end user, end user application and the agent application must terminate in a clean fashion.
3. The server runs continuously and it is therefore OK to terminate it with Ctrl + C option. No points will be deducted for this.
4. Always make sure that you do not have any runaway thread or process when you exit, your CISE account may be suspended by the system admins if you ignore this step.
5. You may pass necessary arguments to start any application either using command line arguments or by using a common configuration file. You may assume that you know the server receiver thread unicast ip address and port number before starting the agent application.
6. All program exceptions and error situations must be handles gracefully and logically.
7. On campus students will be given a 15 minutes demo slot during TA office hours and the grades will be awarded on the spot at the end of the demo period. You can use your laptops for demo purpose.

**Project Requirements:**

Your submission must contain a comprehensive documentation named *Programmer's Guide* explaining your code structure. It should contain a brief writeup of various modules and functions that you use and also explain briefly your design philosophy. It must also contain a separate file named *Installation Guide* containing all necessary instructions for the TA to successfully execute your code. Also follow clean and structured coding style making use of proper indentation and appropriate comments. It is extremely important that your threads terminate properly. Since this is an open specification project, you must also submit a third document *Protocol Specification* specifying the message formats for server and agent communication as well as for end user to agent's command server thread communication and any additional protocol design used in the project.

- Missing Programmer's Guide – 5% penalty
- Missing Protocol Specification – 10% penalty
- Missing Installation Guide – 5% penalty
- Shabby coding style – 5% penalty
- Runaway threads (threads do not terminate on program termination) – 15% penalty

**Submission Instructions:**

You will use UF-IBA submission system to submit your project. Please follow these instructions closely while submitting -

1. Tar all your files including the comprehensive documentation file into a single file using this command – `tar cvf proj1.tar <file list>`
2. Login to UF-IBA system using your login and image password
3. Choose submit homework options from cop5615fa06 control options
4. Submit your tar file and select project 2 from the drop down list
5. Note your confirmation number
6. Late submissions are not encouraged and will result in heavy penalty