Assignment 2 : README

The Code :

server.c

- The serverAddress (sockaddr_in data type) variable is the socket uses a port which has to be specified at the time of execution as parameters.
- The port number is converted from a char* to int.
- The socket() function creates an endpoint for communication and returns a file descriptor that refers to that endpoint (oldsockfd).
- bzero() function function erases the data in serverAddress by writing zeros (bytes containing '\0') to that area.
- Now, the serverAddress structure is initialised. sin_family and sin_port are specified.
- The htons() function converts the unsigned short integer hostshort from host byte order to network byte order.
- bind() assigns the address specified by serverAddress to the socket referred to by the file descriptor oldsockfd. sizeof(serverAddress) specifies the size, in bytes, of the address structure pointed to by serverAddress. Traditionally, this operation is called "assigning a name to a socket".
- listen() marks the socket referred to by oldsockfd as a passive socket,that is, as a socket that will be used to accept incoming connection requests using accept. The second argument (called backlog argument) defines the maximum length to which the queue of pending connections for oldsockfd may grow (kept equal to 7 by me).
- An infinite loop is now initiated which accepts new clients and adds them to a list containing all the clients, allclients. allClients is a structure which contains the file descriptors of all the open sockets.
- The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().
- For every new client, a new thread is being created which is carrying out the receiveMsg() function.
- recv() is used to receive messages from a socket. It is normally used only on a connected socket. A negative return value indicates an error.

- The recv() is implemented in a while() loop which takes the message, processes it, and sends it to required clients.
- A private message will start with a '@' followed by the client number.
- A broadcast message will be written normally.
- The send() call may be used only when the socket is in a connected state (so that the intended recipient is known).
- After sending messages, the buffers are emptied again using bzero(), which has been described earlier.
- The close() will close the file descriptor passed as argument.

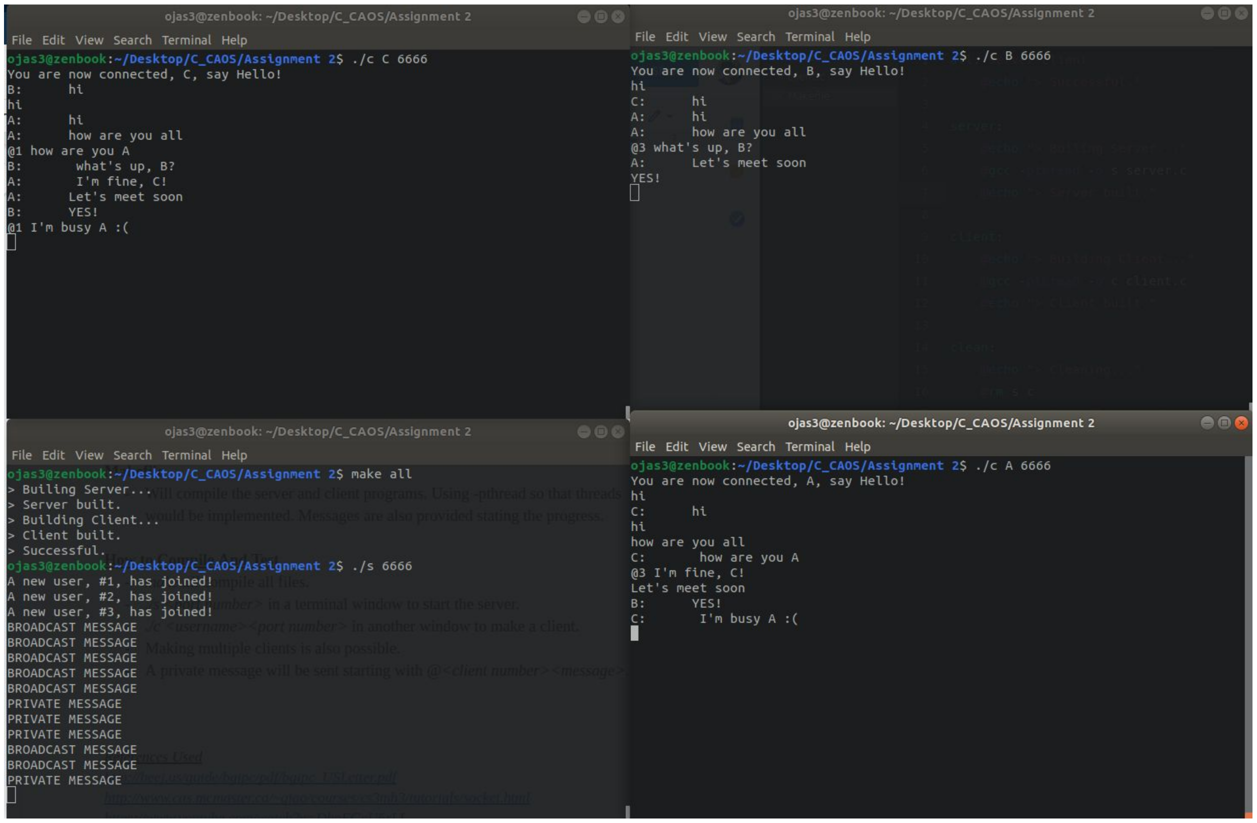
client.c

- The client will have a username which has to be passed as a parameter at the time of execution. A port number (which will be the same as the server port) also has to be specified.
- The socket() function creates an endpoint for communication and returns a file descriptor that refers to that endpoint (oldsockfd).
- bzero() function function erases the data in clientAddress by writing zeros (bytes containing '\0') to that area.
- Now, the clientAddress structure is initialised. sin_family and sin_port are specified. The htons() function converts the unsigned short integer hostshort from host byte order to network byte order.
- The connect() system call connects the socket referred to by the file descriptor oldsocketfd to the address specified by clientAddress. The sizeof(clientAddress) argument specifies the size of clientAddress. The format of the address in clientAddress is determined by the address space of the socket oldsocketfd.
- The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().
- For every new client, a new thread is being created which is carrying out the receiveMsg() function.
- write() writes up to count bytes from the buffer starting at curMsg to the file referred to by the file descriptor oldscoketfd.
- fgets() reads in at most one less than size characters from stream and stores them into the buffer pointed to by buffer. Reading stops after an EOF or a

newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer.
- After writing to the oldsocketfd, the buffers are emptied again using bzero(), which has been described earlier.
- The pthread_join() function waits for the thread specified by thread to terminate.
- The close() will close the file descriptor passed as argument.

Makefile
- Will compile the server and client programs. Using -pthread so that threads would be implemented. Messages are also provided stating the progress.

<u>How to Compile And Test</u>
- *make all* compile all files.
- *./s <port number>* in a terminal window to start the server.
- *./c <username><port number>* in another window to make a client.
- Making multiple clients is also possible.
- A private message will be sent starting with @*<client number><message>*.
- Test screenshot has been provided :-



ojas3@zenbook: ~/Desktop/C_CAOS/Assignment 2
File Edit View Search Terminal Help

```
ojas3@zenbook:~/Desktop/C_CAOS/Assignment 2$ ./c C 6666
You are now connected, C, say Hello!
B:      hi
hi
A:      hi
A:      how are you all
@1 how are you A
B:       what's up, B?
A:       I'm fine, C!
A:      Let's meet soon
B:      YES!
@1 I'm busy A :(
```

```
ojas3@zenbook:~/Desktop/C_CAOS/Assignment 2$ ./c B 6666
You are now connected, B, say Hello!
hi
C:      hi
A:      hi
A:      how are you all
@3 what's up, B?
A:      Let's meet soon
YES!
```

```
ojas3@zenbook:~/Desktop/C_CAOS/Assignment 2$ make all
> Builing Server...
> Server built.
> Building Client...
> Client built.
> Successful.
ojas3@zenbook:~/Desktop/C_CAOS/Assignment 2$ ./s 6666
A new user, #1, has joined!
A new user, #2, has joined!
A new user, #3, has joined!
BROADCAST MESSAGE
BROADCAST MESSAGE
BROADCAST MESSAGE
BROADCAST MESSAGE
BROADCAST MESSAGE
PRIVATE MESSAGE
PRIVATE MESSAGE
PRIVATE MESSAGE
BROADCAST MESSAGE
BROADCAST MESSAGE
PRIVATE MESSAGE
```

```
ojas3@zenbook:~/Desktop/C_CAOS/Assignment 2$ ./c A 6666
You are now connected, A, say Hello!
hi
C:      hi
hi
how are you all
C:       how are you A
@3 I'm fine, C!
Let's meet soon
B:      YES!
C:       I'm busy A :(
```

Errors

- In **server.c**, errors are flagged when :
    - send() function fails, i.e. an error occurs while sending a message to a particular client. This might happen when the client is no longer connected to the server.
    - Binding error occurs when another program is using the port which we have passed as a parameter.
    - accept() function gives an error if The socket is marked non-blocking and no connections are present to be accepted. An error is flagged if the oldsockfd is not an open file descriptor or doesn't refer to a socket. An error may also be flagged when a connection has been aborted.
- In **client.c**, errors are flagged when :
    - connect() error when `The user tried to connect to a` broadcast address without having the socket broadcast flag enabled or the connection request failed. A non-blocking socket will also result in an error. An invalid file descriptor will also result in an error. Errors will also be flagged if the socket is already connected, or when there is a timeout while attempting connection.
    - write() will flag an error if the oldsocketfd is unvalid.

- If the user tries to send a private message to an unavailable client, an Input Error is flagged.
- An Input Error will also be flagged if the message is empty and contains nothing.

References Used

http://beej.us/guide/bgipc/pdf/bgipc_USLetter.pdf
http://www.cas.mcmaster.ca/~qiao/courses/cs3mh3/tutorials/socket.html
https://www.youtube.com/watch?v=DboEGcU6rLI
http://man7.org/linux/man-pages/index.html
https://www.cs.cmu.edu/~srini/15-441/S10/lectures/r01-sockets.pdf
https://www.usna.edu/Users/cs/aviv/classes/ic221/s16/lec/26/lec.html