To implement the program I have created some modules to solve it. There are some sections on the problem. They are:

* Register peer
* Handshake message and handshake acknowledgment
* Data packet sending

This three sections must be done for both sender and receiver client. In my implementation I first decide which one act as a sender or receiver. This decisions is based on the command line arguments. After this, the task of register peer comes. At first the receiver peer registered with the procedure *registeredrec()*. And as the same manner the sender peer also register with the procedure *registeredsen()*
After that the sender peer send a handshake message send to receiver peer forwarded by server. This is done with the help of *handshaking()* procedure. As a manner the receiver receives handshaking message from the sender and issues a handshaking acknowledgment message to the sender with the *handshakeack()* function. So this is the first two parts on the assignment. A bit easy section to do.

The most critical section is data transfer section. In a packet maximum 999 bytes of data can be send. So if the file size is greater than 996 bytes we have to send segmented packets to send the contains of the file. After sending a data packet the sender must be wait for the acknowledge from the receiver side. After getting the acknowledge it sends the next possible data packet to the receiver. *sendfile()* and *recfile()* is responsible to accomplish it. To track in the file which I want to send I use *fseek()* system call. Though this can be done without *fseek()* but it is natural to control random access on a file with *fseek()*.

I faced some problem on the data message implementation section. At first, the server can't forward the data message to the sender it gives bad address error. To solve the problem, I change the way of sending the data size on the packet. I changed the network byte order to the host byte order with *ntohs()* system call. And after that the first problem has gone. The second one is much peculiar. After sending two data packets successfully to the sender the length of the file done to zero and no more data packets are sent though there remains data. I have not faced this problem on my own machine which is run on Ubuntu 8.04. But when I check my code at department I discover this. To solve this I do some tricky programming. At last, I successfully solve the problem and still in confused when it does not work at Fedora but fairly work at Ubuntu.

After sending all the data message between sender and receiver it is the time to close the sockets which we create and bind on the port. This is the formal termination of our code. But if the code faces any problem or errors it immediately exit.

So it is the overall structure of my code. I formally give the prototype here for better understanding.

```
int issender(int argc, char **argv);
int senderpeer(int argc, char **argv , PEEROPS *senops);
int receiverpeer(int argc, char **argv, PEEROPS *recops);
int registeredrec(DATAGRAM *recpack, PEEROPS *recops);
int registeredsen(DATAGRAM *senpack, PEEROPS *recops);
int handshaking(PEEROPS *recops);
int handshakeack(PEEROPS *senops);
int checkerror(DATAGRAM *error);
int recfile(PEEROPS *senops);
int sendfile(PEEROPS *recops)
```

Other than the above procedures I also create two structures to hold peer options and packet header and data message. I reuse the RDTHEADER structure with the extension of holding data message of the maximum 996 bytes. This one is a static array of 996 size to hold the data message.

So this is overall of the code. This program can run on different version of Fedora and Ubuntu, but I test on Fedora 4 and Ubuntu 8.04.

The whole program is coded by me. No part of the code taken from any sources including hard copy or from Internet by goggling. But the overall idea is taken from the provided code. This code heps me much to do the job correctly.

Further improvement can be done on the code. From the command line section this can be do like gui programming with java. I will try to make it gui based on future upon free time. By giving further documentation this program may be much enriched in future.