1. Introduction

The final project for COE768 Computer Networks consists of creating a simple peer to peer network. This project serves as the culmination of everything that was taught throughout the course via laboratory assignments. A peer to peer network application is an infrastructure where multiple computer systems connect and share resources without the involvement of a designated centralized server. The infrastructure that was used for this final project consisted of multiple peers that transferred content amongst each other with the help of an index server keeping track of all interactions.

1.1 Socket programming, TCP and UDP Explained

However, in order to explain how this application was implemented, it is important to first go over the basics of what socket programming is. A socket is essentially an endpoint of a two-way communication between two computer systems on a network. It is usually bound to a port number and IP address so the transport layer can identify where the data needs to be sent to. It lies as an interface between the transport layer and application layer within the OSI Model. Socket programming is when sockets are created and manipulated to create desirable applications that deal with multiple processes communicating over a network.

For example, a simple TCP connection was established in lab 2 & 3 using socket programming.

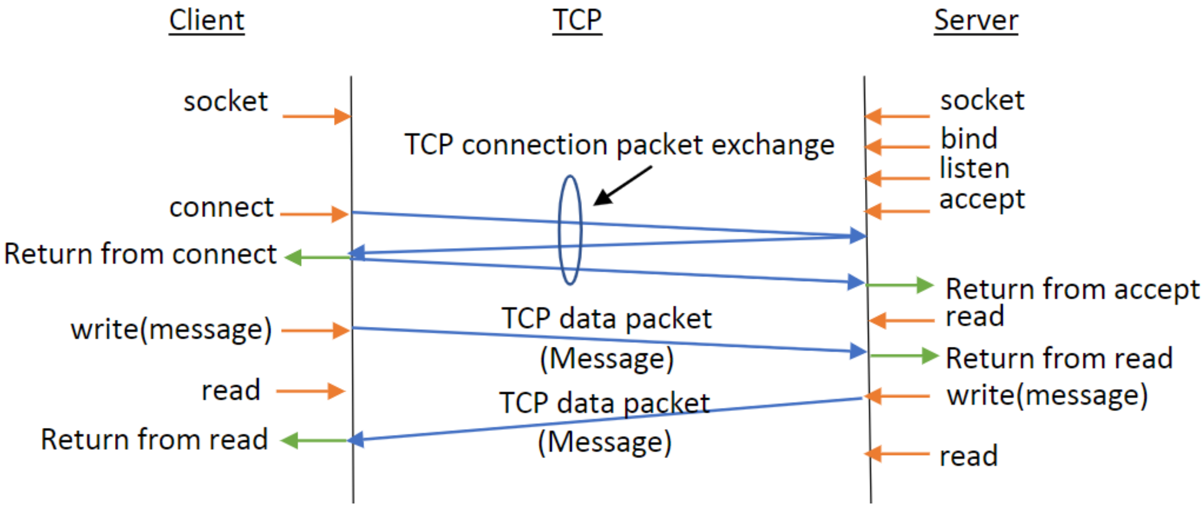


Figure 1

As seen above in figure 1, one socket is bound to the server and enters a passive mode where it simply listens for any connection requests. Once the client socket reaches out to form a connection, a TCP connection is established where both parties can communicate using read() and write() methods.

Another transport connection protocol is the UDP protocol which was implemented in lab 5. The difference between UDP and TCP is that UDP does not require an established connection to be established in order to send data. This connectionless nature of UDP makes it less reliable, whereas TCP is reliable and guaranteed because of being connection-oriented.

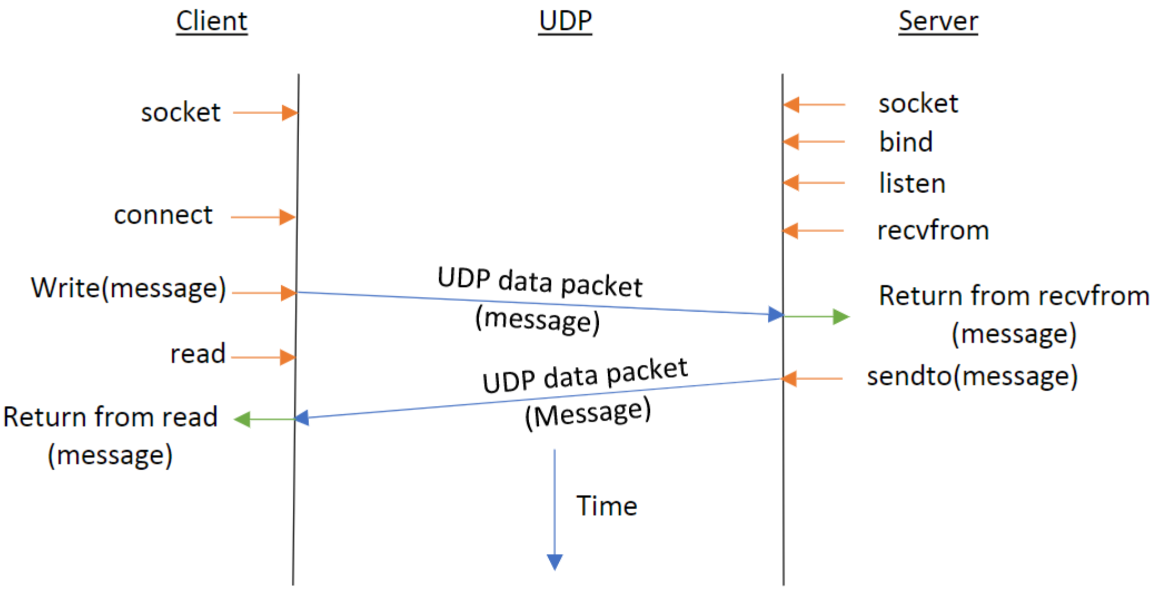

Client
UDP
Server
socket
connect
Write(message)
read
Return from read (message)
socket
bind
listen
recvfrom
UDP data packet (message)
Return from recvfrom (message)
sendto(message)
UDP data packet (Message)
Time

Figure 2

As seen above in figure 2, the connection does not need to be established between the client and server. The client can simply send the message without confirming if the server has accepted the connection request.

2. Description of the Client and Server Programs

2.1 Basic Description of P2P Mechanism

The basic premise of the peer-to-peer application is that each peer can act as a client and a server depending on the situation. Supposing a peer has a piece of content that it wants to make available to the other peers on a network. It would register the content onto an index server. The index server holds a list of all pieces of content and their respective content-servers that are a part of the peer-to-peer server. While the content is being registered, the peer opens a passive TCP socket for the purposes of transferring file data. The system stays on idle until a 2nd peer requests for content download. If the content is registered on the content list in the index server, then the index server sends the relevant information needed to establish connection between the 2 peers. Once the content-client peer receives the address of the content-server peer, it reaches out and establishes a TCP connection. The two peers then begin to exchange data packets in order to transfer the content from the content-server to the content-peer. If a peer ever wants to remove itself from the network, it can simply let the index server know, and the index server will proceed to deregister all of the pieces of content that were under that peer’s name.

All interactions between index-server and peers happen via UDP protocol. The actual downloading between two peers happens via TCP protocol.

2.1.1 PDUs Explained

A PDU (protocol data unit) is a data structure used throughout the project to exchange information between the index server and peers. It consists of a “Type” field (1 byte), and a “Data” field. The “Type” field specifies what type of PDU it will be, as different PDU types will have different purposes and slightly different content. The “Data” field contains the actual data that is to be transferred.

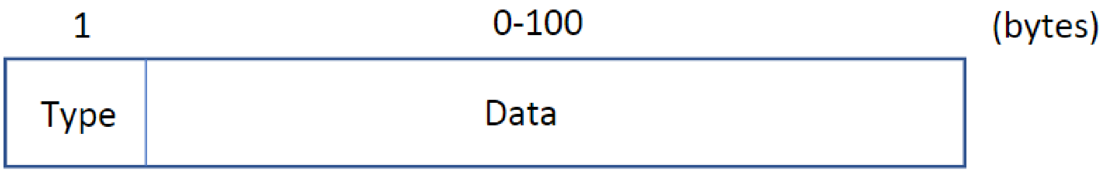


Figure 3

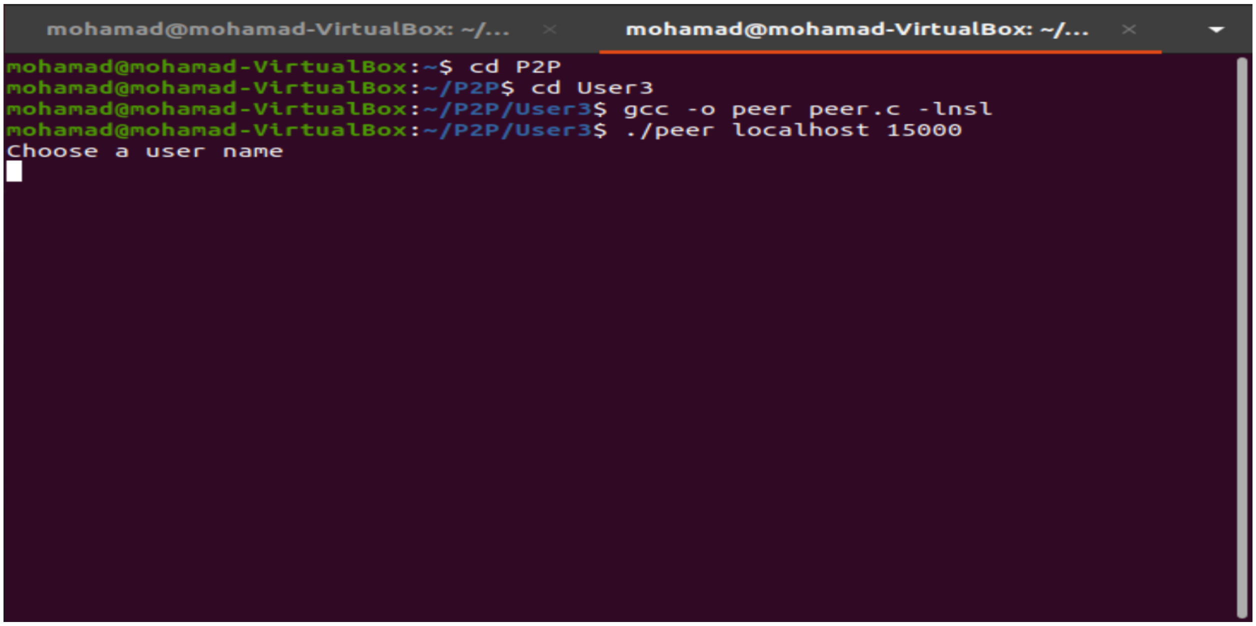
For the project, there are a total of eight PDU types that were used. The table below explains all the various types, alongside their structure and description.

Type	**Structure**	**Description**
R Registration	Type(R) Peer name Content name Address & Port	- Sent from peer to index - When peer has a piece of content that it wants to register onto the network
D Download	Type (D) Content Name	- Sent from Content client to Content Server - This is to initiate the download sequence between the 2 peers
S Search	Type (S) Content name	- Sent between both peer and index server - If peer wants to search for a certain content and it’s respective content server, it sends this to index-server to check if it contains it in the list - If server has this content/peer registered, it sends back another PDU to the peer with the address attached as well
T	Type (T)	- Sent from Peer to Index Server

De-Registration	Peer name Content name	- If the peer wants to remove a specific piece of content from the list, then it sends this PDU to make a deregistration request
C Content Data	Type (C) Data	- Content server to Content Client - This PDU carries over the actually data during the downloading process between the two peers
O List of Online Registered content	Type (O)	- Sent between peer and index server - When the peer would like to receive an entire list of everything that’s been registered on the P2P network - Index sends a list of all items on the list
A Acknowledgment Message	Type(A)	- Sent from Index server to peer
E Error Message	Type(E)	- This PDU can be used anywhere whenever there is an error between to nodes

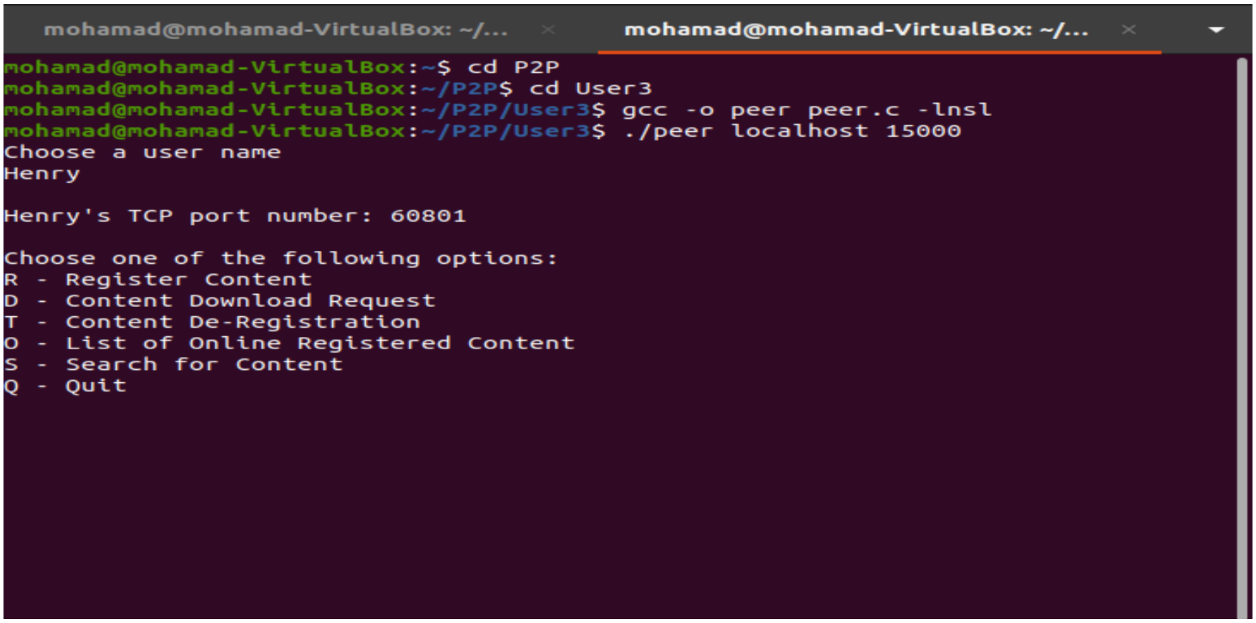
3. Observation and Analysis

3.1 Demonstration of Code VIA Screenshots


mohamad@mohamad-VirtualBox: ~/...
mohamad@mohamad-VirtualBox: ~/...

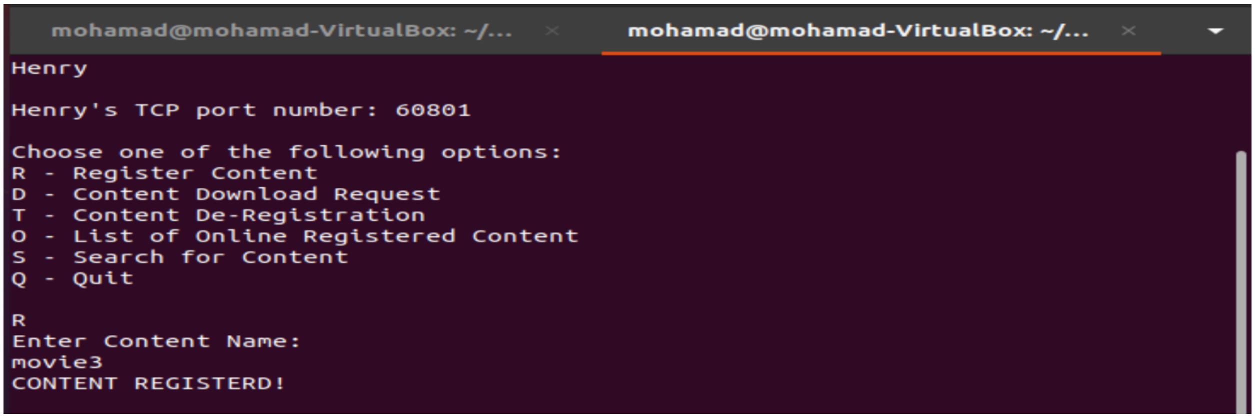
```
mohamad@mohamad-VirtualBox:~$ cd P2P
mohamad@mohamad-VirtualBox:~/P2P$ cd User3
mohamad@mohamad-VirtualBox:~/P2P/User3$ gcc -o peer peer.c -lnsl
mohamad@mohamad-VirtualBox:~/P2P/User3$ ./peer localhost 15000
Choose a user name
```

Figure 4:The entrance screen for peers


mohamad@mohamad-VirtualBox: ~/...
mohamad@mohamad-VirtualBox: ~/...

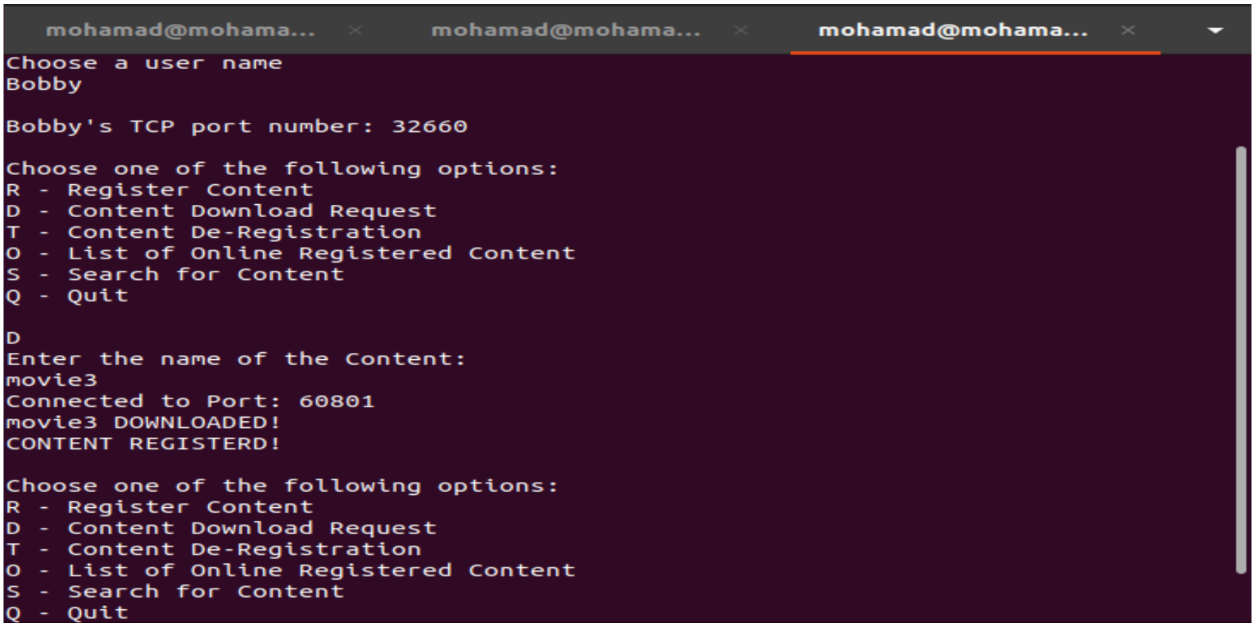
```
mohamad@mohamad-VirtualBox:~$ cd P2P
mohamad@mohamad-VirtualBox:~/P2P$ cd User3
mohamad@mohamad-VirtualBox:~/P2P/User3$ gcc -o peer peer.c -lnsl
mohamad@mohamad-VirtualBox:~/P2P/User3$ ./peer localhost 15000
Choose a user name
Henry

Henry's TCP port number: 60801

Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit
```

Figure 4: When the peer choses a user name


mohamad@mohamad-VirtualBox: ~/...
mohamad@mohamad-VirtualBox: ~/...

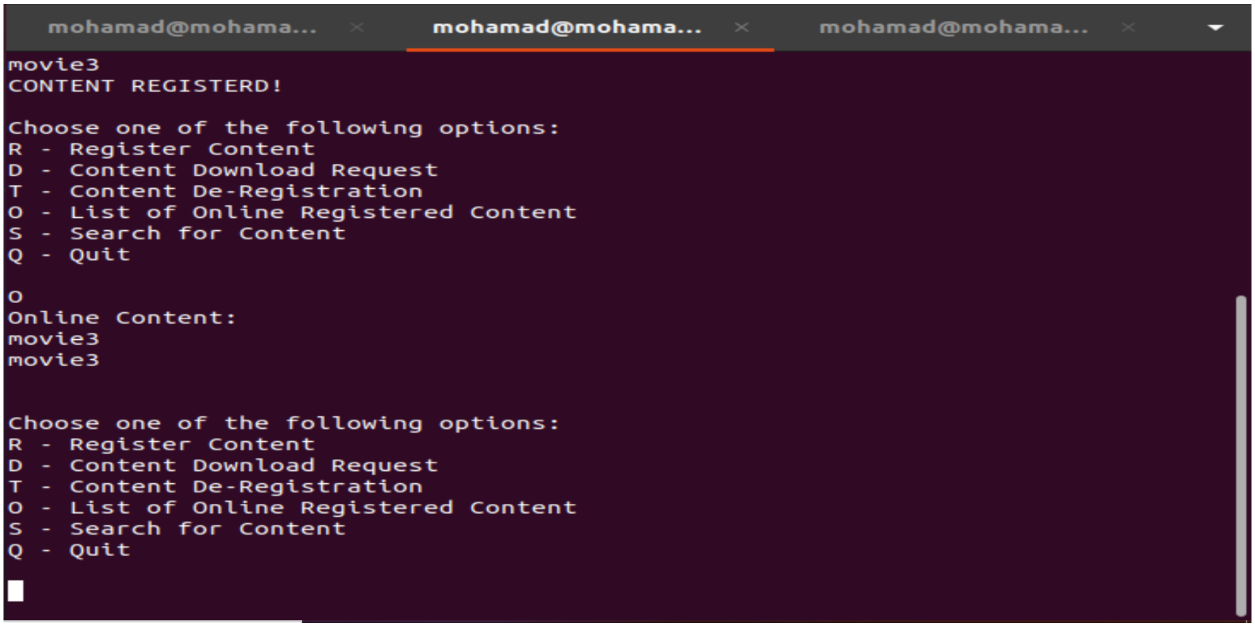
```
Henry

Henry's TCP port number: 60801

Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit

R
Enter Content Name:
movie3
CONTENT REGISTERD!
```

Figure 5: When the peer choses to register a file



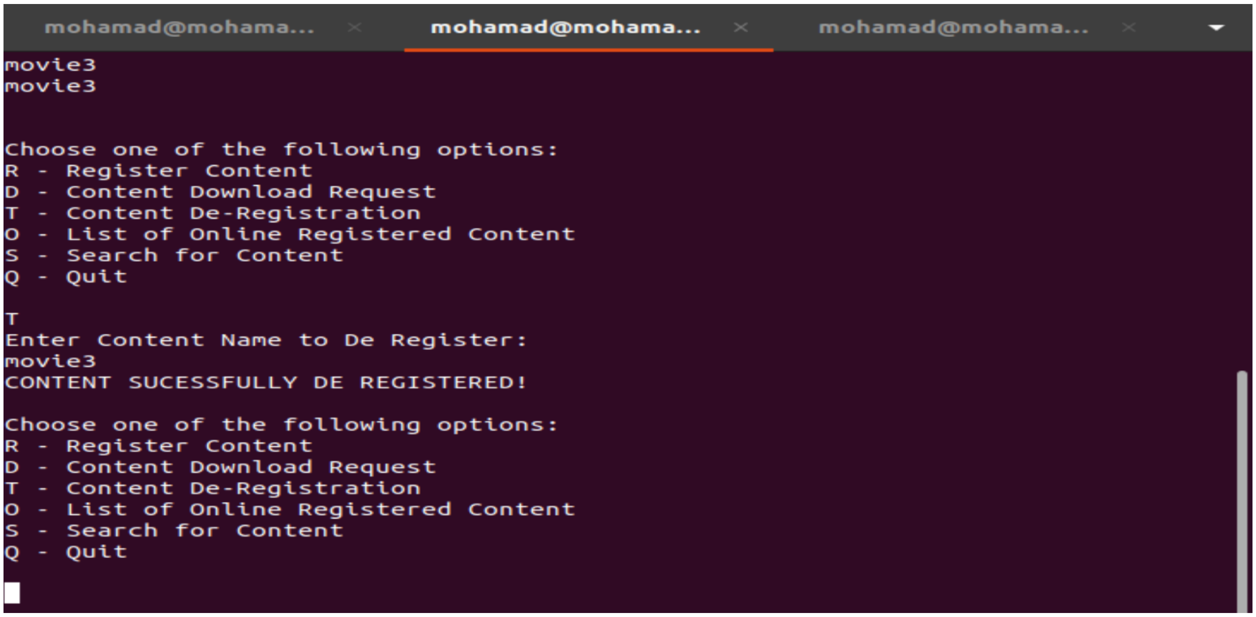
```
Choose a user name
Bobby

Bobby's TCP port number: 32660

Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit

D
Enter the name of the Content:
movie3
Connected to Port: 60801
movie3 DOWNLOADED!
CONTENT REGISTERD!

Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit
```

Figure 6: When the peer choses to download a file



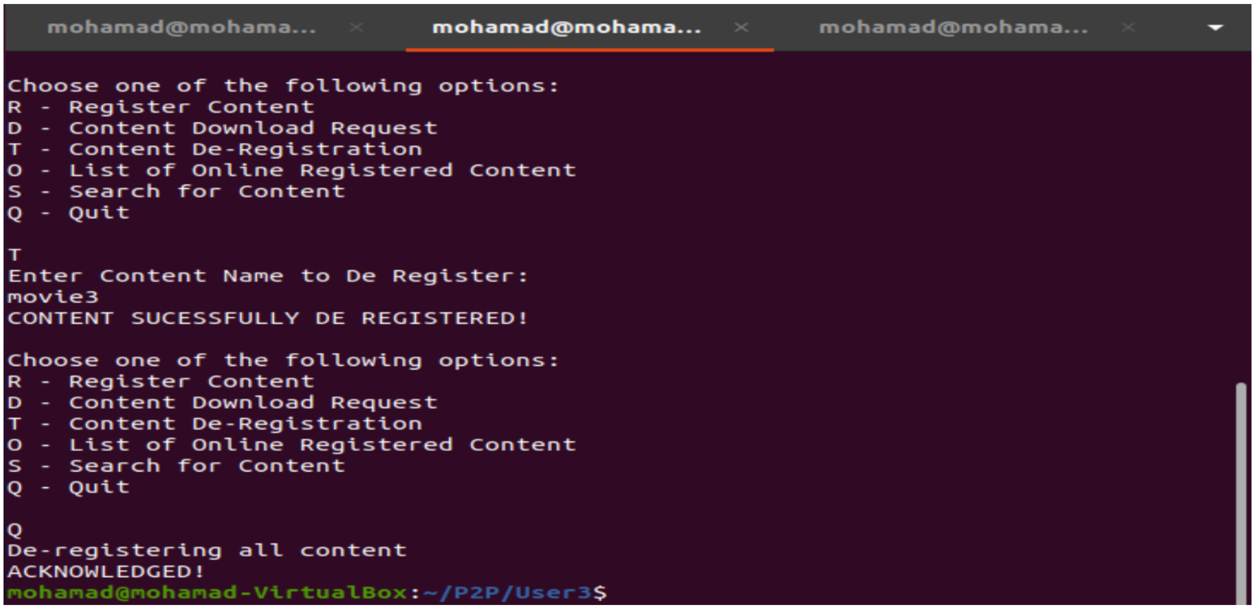
```
movie3
CONTENT REGISTERD!

Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit

O
Online Content:
movie3
movie3


Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit
```

Figure 7: When the peer choses to list all online content



```
movie3
movie3


Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit

T
Enter Content Name to De Register:
movie3
CONTENT SUCESSFULLY DE REGISTERED!

Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit
```

Figure 8: When the peer choses to de-register a file



```
Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit

T
Enter Content Name to De Register:
movie3
CONTENT SUCESSFULLY DE REGISTERED!

Choose one of the following options:
R - Register Content
D - Content Download Request
T - Content De-Registration
O - List of Online Registered Content
S - Search for Content
Q - Quit

Q
De-registering all content
ACKNOWLEDGED!
mohamad@mohamad-VirtualBox:~/P2P/User3$
```

Figure 9: When the peer choses to quit

4. Overview of Main Functions of the Code

Register Function:

In order to register a content, we ask the user to enter the content name. Then, we use this along with the user name and address/port of the user to send an RPDU in string format to the index server. Upon receiving, the index server converts the string to RPDU struct format and uses the information to check if the data array on its side contains a matching peer name and content name combination. If it does exist, the index server returns an error PDU to the peer and the peer notifies the user to pick a new peer name. If the combination doesn't exist, the index server adds the data (peer name, content name and address) to the next index of the data array and then returns an acknowledgment PDU back to the client. Once the client receives the acknowledgment PDU, it notifies the user that the content was successfully registered.

Deregister Function:

In order to de-register a content, we ask the user to enter the content name. Then, we use this along with the user name to send a TPDU in string format to the index server. Upon receiving, the index server converts the string to TPDU struct format and uses the information to check if the data array on its side contains a matching peer name and content name combination. If it does exist, it deletes the data in the corresponding index of the data array by zeroing it out. Then it sends an acknowledgment PDU to the peer to notify it. Once receiving the acknowledgement PDU, the peer prints an acknowledgment to the user to notify that the content was successfully deleted. If the combination doesn't exist in the data array, the index server sends an Error PDU to the peer. Once receiving the error PDU, the peer notifies the user that the content does not exist so it wasn't de-registered.

Content Download:
Once the program receives an SPDU from the index to the peer with the information of the content-server, it starts to set a TCP connection with the server. It does so by first emptying out the server structure and copying the host IP in. It then creates a system socket using the same functions and procedures used in previous labs of the course. On the other side, the peer is simply in a passive listening mode, waiting for a TCP connection request to form. Once both sides establish the TCP connection, the usual downloading proceeds with a file being sent through different packets over the transport layer. Once it successfully downloads, the peer automatically registers the newly downloaded content onto the P2P network.

List:
If the user wants to see the list of all the content that is currently registered on the P2P network, they enter “O” into the options menu. A PDU is sent to the index, and a subsequent string is sent back from the index server containing the entire list of registered pieces of content. This is done by taking the content names from the array and lining them up into one string. The entire list is then simply printed out using Printf.

Search:
If the user wants to search a specific content that was registered, the program would ask the user to enter the name of the desired content. Once it saves that name onto a pdu, it sends it over to the index server. After receiving the name, the index then uses findContent() to check if the content is actually available within the stored array. If there is, the function confirms it and the index server sends over the name and port number back to the peer through a pdu.

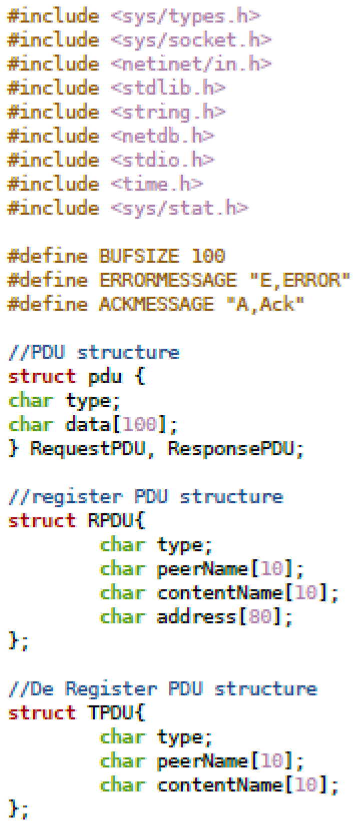
Quit:
In order to end the program, the user has the option to quit it entirely. This will cause the peer to fully deregister all pieces of content from the P2P network. This is done so by sending a Qpdu to the index server. Once it receives the notion to register all pieces of content, it iterates through the list and clears all areas that have that specific peer/port saved. Once clears it all, it sends the acknowledgement.

5. Conclusions

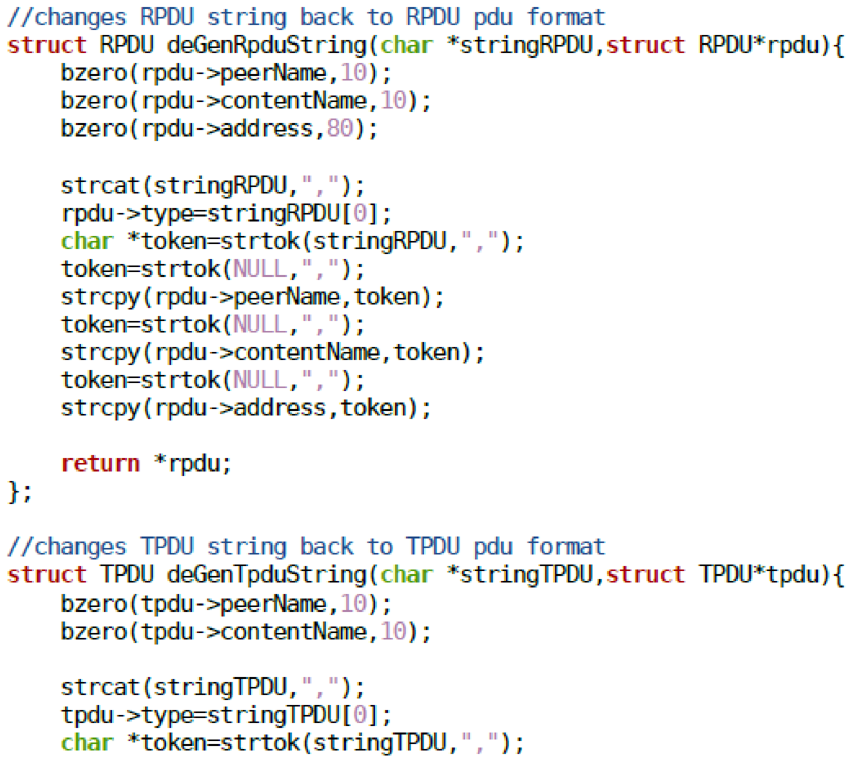
In conclusion, the peer to peer application was able to host multiple computer systems as active members, each being able to act as both content-peer-client and content-peer-server. The peers were able to successfully download content from one end to another through the TCP protocol. Each peer was also successfully able to register content, de-register content, request content provider address, view all available content and lastly quit. The index server was able to manage all interactions with the peers using UDP protocol, and it was able to hold a list of all registered pieces of content that were a part of the P2P network. Therefore, our application met all the specified requirements and functions as it should.

6. Appendix

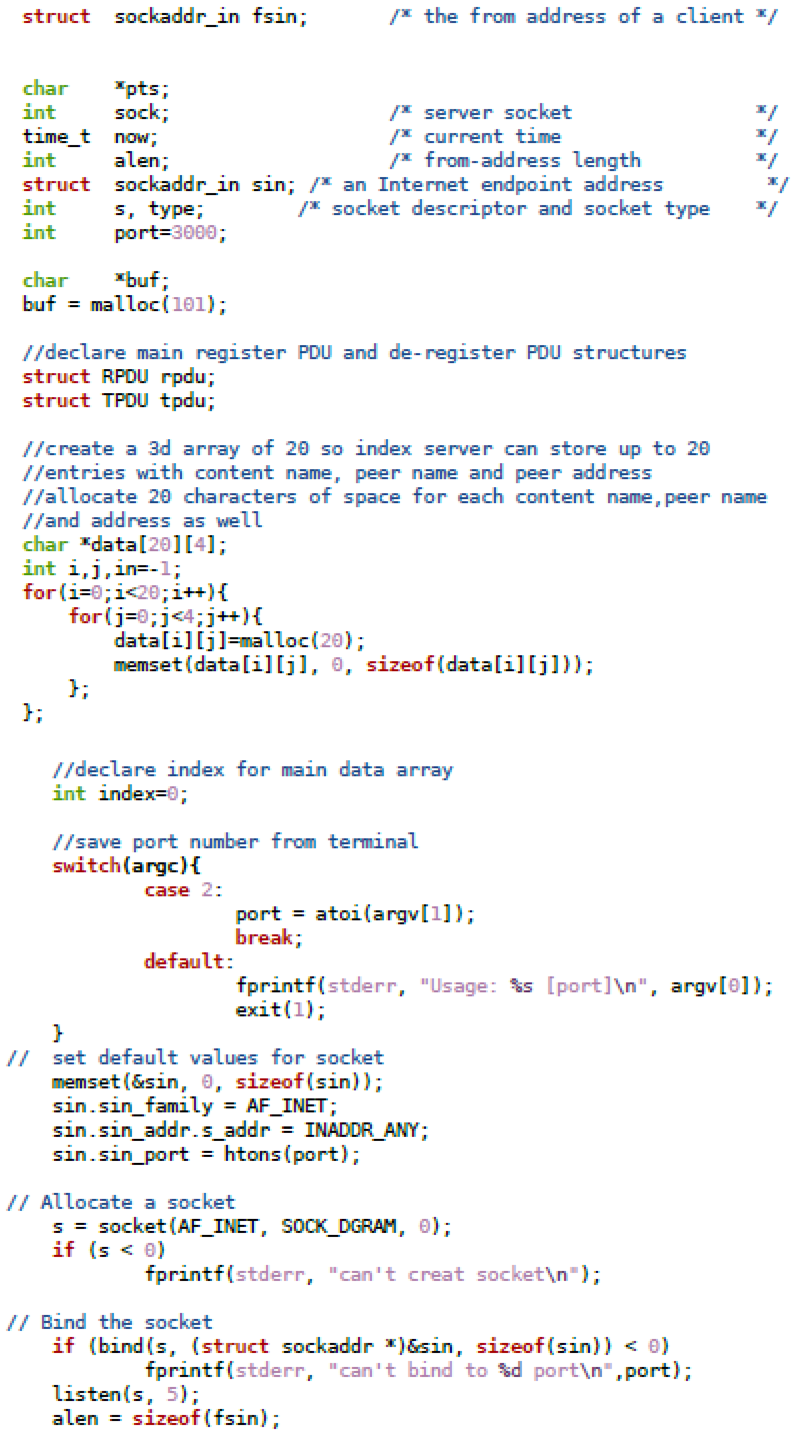
Index Server Code



```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <stdio.h>
#include <time.h>
#include <sys/stat.h>

#define BUFSIZE 100
#define ERRORMESSAGE "E,ERROR"
#define ACKMESSAGE "A,Ack"

//PDU structure
struct pdu {
char type;
char data[100];
} RequestPDU, ResponsePDU;

//register PDU structure
struct RPDU{
        char type;
        char peerName[10];
        char contentName[10];
        char address[80];
};

//De Register PDU structure
struct TPDU{
        char type;
        char peerName[10];
        char contentName[10];
};
```



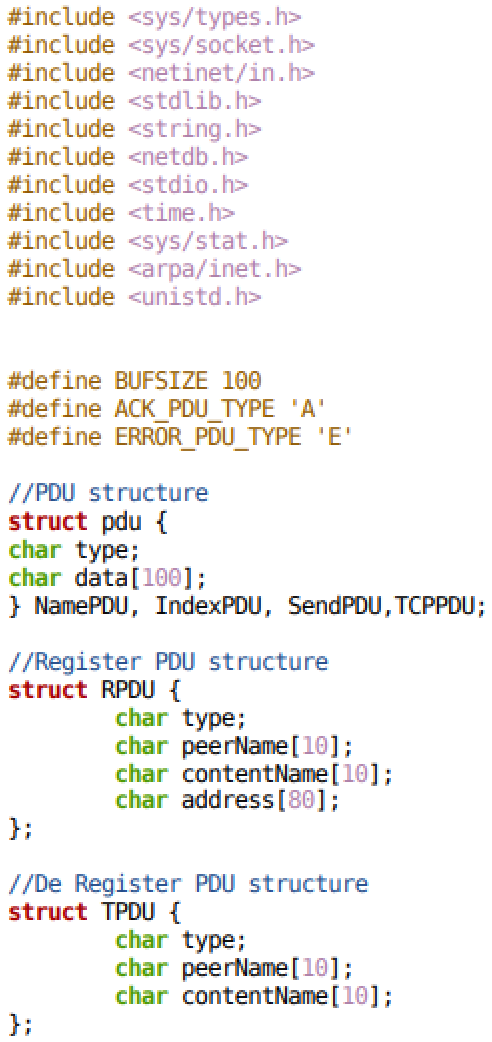
```
//changes RPDU string back to RPDU pdu format
struct RPDU deGenRpduString(char *stringRPDU,struct RPDU*rpdu){
    bzero(rpdu->peerName,10);
    bzero(rpdu->contentName,10);
    bzero(rpdu->address,80);

    strcat(stringRPDU,",");
    rpdu->type=stringRPDU[0];
    char *token=strtok(stringRPDU,",");
    token=strtok(NULL,",");
    strcpy(rpdu->peerName,token);
    token=strtok(NULL,",");
    strcpy(rpdu->contentName,token);
    token=strtok(NULL,",");
    strcpy(rpdu->address,token);

    return *rpdu;
};

//changes TPDU string back to TPDU pdu format
struct TPDU deGenTpduString(char *stringTPDU,struct TPDU*tpdu){
    bzero(tpdu->peerName,10);
    bzero(tpdu->contentName,10);

    strcat(stringTPDU,",");
    tpdu->type=stringTPDU[0];
    char *token=strtok(stringTPDU,",");
    token=strtok(NULL,",");
    strcpy(tpdu->peerName,token);
    token=strtok(NULL,",");
    strcpy(tpdu->contentName,token);

    return *tpdu;
};


//check to see if content is registered and return its index
int findDContent(char *data[20][4],char*contentName){
    int i;
    int index=-1;
    for(i=0;i<20;i++){
        if((strcmp(contentName,data[i][1])==0)&&index==-1){
            index=i;
        }
        else if(index!=-1&&(strcmp(contentName,data[i][1])==0)){
            if(atoi(data[i][3])<atoi(data[index][3])){
                index=i;
            }
        }
    }
    return(index);
};


//check if there is an existing content name matching to peer name
//in the data array, if there is returns 0, if unique then returns 1
int checkData(char *data[20][4],char*peerName,char*contentName){
    int i;
    int flag=1;
    for(i=0;i<20;i++){
        if((strcmp(contentName,data[i][1])==0)&&(strcmp(peerName,data[i][0])==0)){
            flag=0;
        }
    }
    return(flag);
};

 //searches data array to see if there is an entry that matches the
 //provided peerName and contentName, if there is it returns the index
 //otherwise it returns -1
 int findContent(char *data[20][4],char*peerName,char*contentName){
     int i;
     int index=-1;
     for(i=0;i<20;i++){
         if((strcmp(contentName,data[i][1])==0)&&(strcmp(peerName,data[i][0])==0)){
             index=i;
         }
     }
     return(index);
 };


 /*------------------------------------------------------------------------
  * main - Iterative UDP server for TIME service
  *------------------------------------------------------------------------
  */
 int
 main(int argc, char *argv[])
 {
```



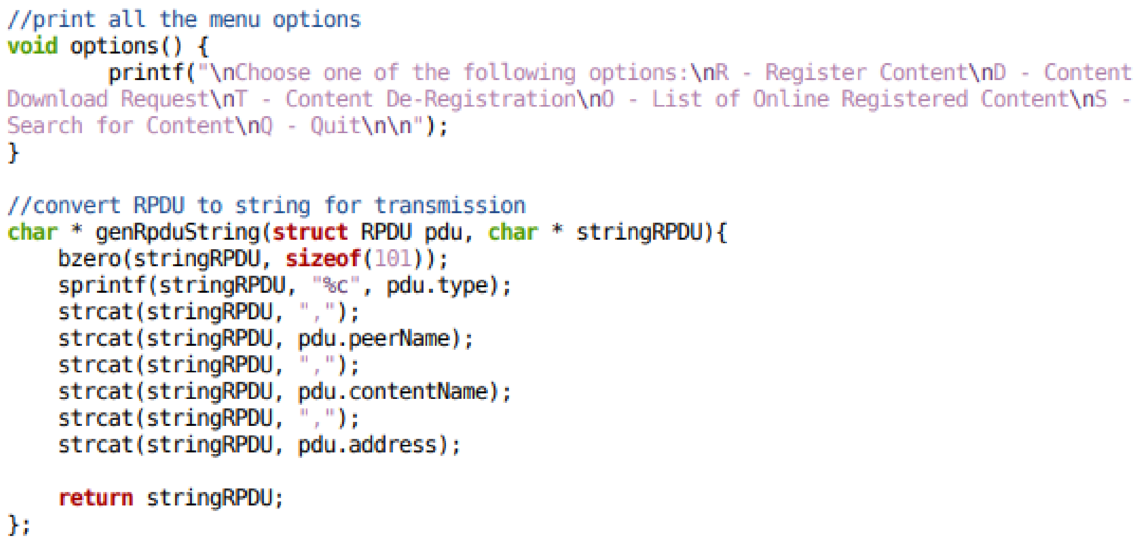
```
 struct  sockaddr_in fsin;       /* the from address of a client */


 char    *pts;
 int     sock;                   /* server socket                */
 time_t  now;                    /* current time                 */
 int     alen;                   /* from-address length          */
 struct  sockaddr_in sin; /* an Internet endpoint address         */
 int     s, type;        /* socket descriptor and socket type    */
 int     port=3000;

 char    *buf;
 buf = malloc(101);

 //declare main register PDU and de-register PDU structures
 struct RPDU rpdu;
 struct TPDU tpdu;

 //create a 3d array of 20 so index server can store up to 20
 //entries with content name, peer name and peer address
 //allocate 20 characters of space for each content name,peer name
 //and address as well
 char *data[20][4];
 int i,j,in=-1;
 for(i=0;i<20;i++){
     for(j=0;j<4;j++){
         data[i][j]=malloc(20);
         memset(data[i][j], 0, sizeof(data[i][j]));
     };
 };

    //declare index for main data array
    int index=0;

    //save port number from terminal
    switch(argc){
            case 2:
                    port = atoi(argv[1]);
                    break;
            default:
                    fprintf(stderr, "Usage: %s [port]\n", argv[0]);
                    exit(1);
    }
//  set default values for socket
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);

// Allocate a socket
    s = socket(AF_INET, SOCK_DGRAM, 0);
    if (s < 0)
            fprintf(stderr, "can't creat socket\n");

// Bind the socket
    if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
            fprintf(stderr, "can't bind to %d port\n",port);
    listen(s, 5);
    alen = sizeof(fsin);
```

```
        char pdu_type;
        char BUFF[101];
        memset(BUFF,0,strlen(BUFF));
        FILE * fp;
        struct stat st;
        int size;

        while (1) {
                memset(&BUFF, 0 , sizeof(BUFF));
                //receiving message from peer:
                recvfrom(s, BUFF, BUFSIZE, 0,(struct sockaddr *)&fsin, &alen);
                strcpy(buf,BUFF);
                fflush(stdout);

                memset(&RequestPDU.data, 0 , sizeof(RequestPDU.data));
                //coping the contents of data buffer into RequestPDU
                RequestPDU.type = BUFF[0];
                for (int i = 0; i < BUFSIZE; i++) {
                        RequestPDU.data[i] = BUFF[i + 1];
                }
                switch(RequestPDU.type){
                case 'S':

                        printf("Searching for %s\n", RequestPDU.data);
                        //check registered content to see if
                        //requested content is present
                        //and if it is send back port number
                        strcat(RequestPDU.data,"\n");
                        in =findDContent(data,RequestPDU.data);
                        if( in != -1){
                                memset(&ResponsePDU.data, 0 , sizeof(ResponsePDU.data));
                                ResponsePDU.type = 'S';
                                strcpy(ResponsePDU.data, data[in][2]);
                                sendto(s, &ResponsePDU, strlen(ResponsePDU.data)+1, 0,(struct
sockaddr *)&fsin, sizeof(fsin));
                                //increment entry usage by 1 after each time port details are
requested
                                int val=atoi(data[in][3]);
                                val++;
                                sprintf(data[in][3],"%d",val);
                                }


                  //if the content isnt found then there is an error
                          else {
                                  ResponsePDU.type = 'E';
                                  strcpy(ResponsePDU.data, "File not found");
                                  sendto(s, &ResponsePDU, strlen(ResponsePDU.data)+1, 0,(struct
  sockaddr *)&fsin, sizeof(fsin));
                          }
                          break;
                  case 'O':
                          memset(&ResponsePDU.data, 0 , sizeof(ResponsePDU.data));
                          ResponsePDU.type = 'O';
                          printf("Sending Online-content list\n");
                          // check if there are any content stored and
                          // copy into responsePDU to send back to peer
                          for(int i=0; i<20;i++){
                                  if( strcmp(data[i][1],"")!=0 ){
                                          strcat(ResponsePDU.data, data[i][1]);
                                  }
                          }
                          //check if anything was added to ResponsePDU
                        if( strcmp(ResponsePDU.data,"")==0 ){
                                strcpy(ResponsePDU.data, "Nothing is stored yet");
                        }
                        //send data to peer through socket

                        sendto(s, &ResponsePDU, strlen(ResponsePDU.data)+1, 0,(struct
sockaddr *)&fsin, sizeof(fsin));


                        break;
                case 'R':
                         //convert received string into RPDU format
                        rpdu=deGenRpduString(buf,&rpdu);
                        strcat(rpdu.contentName,"\n");
                        printf("%s requesting to register
%s\n",rpdu.peerName,rpdu.contentName);
                        //check if recieved peer and content name combo is in data
                        //array already, if not allow to send ACK
                        if (checkData(data,rpdu.peerName,rpdu.contentName)==1){
                            sendto(s, ACKMESSAGE, strlen(ACKMESSAGE), 0, (struct sockaddr *)
&fsin, sizeof(fsin));

                            //write peerName,contentName and address to array
                            memcpy(data[index][0],rpdu.peerName,sizeof(rpdu.peerName));
                            memcpy(data[index][1],rpdu.contentName,sizeof(rpdu.contentName));
                            memcpy(data[index][2],rpdu.address,sizeof(rpdu.address));

                            //increment index so next registered content is
                            //put into next array row
                            index++;
                         }


                           //if received name combo is in array already, send ERROR
                           else{
                               sendto(s, ERRORMESSAGE, strlen(ERRORMESSAGE), 0, (struct
  sockaddr *) &fsin, sizeof(fsin));

                           }

                          break;
                  case 'T':
                          //convert received string into TPDU structure
                          tpdu=deGenTpduString(buf,&tpdu);
                          strcat(tpdu.contentName,"\n");
                          printf("%s requesting to de-register
  %s\n",tpdu.peerName,tpdu.contentName);
                          //index to delete
                          int dIndex=findContent(data,tpdu.peerName,tpdu.contentName);
                          //if content is not found in array, return ERROR
                          if(dIndex==-1){
                              sendto(s, ERRORMESSAGE, strlen(ERRORMESSAGE), 0, (struct sockaddr
  *) &fsin, sizeof(fsin));
                          }
                          //if content is found in array, return ACK and zero out
                          //the entry in data array
                          else{
                              bzero(data[dIndex][0],sizeof(data[dIndex][0]));
                              bzero(data[dIndex][1],sizeof(data[dIndex][1]));
                              bzero(data[dIndex][2],sizeof(data[dIndex][2]));
                              sendto(s, ACKMESSAGE, strlen(ACKMESSAGE), 0, (struct sockaddr *)
  &fsin, sizeof(fsin));
                          }
                        break;

                        case 'Q':
                        printf("%s Quiting\n",RequestPDU.data);
                        // check if there are any content stored and
                        // de register them all
                        for(int i=0; i<20;i++){
                                if( strcmp(data[i][1],"")!=0 && strcmp(data[i]-
[0],RequestPDU.data)==0){
                                        bzero(data[i][0],sizeof(data[i][0]));
                                        bzero(data[i][1],sizeof(data[i][1]));
                                        bzero(data[i][2],sizeof(data[i][2]));
                                }
                        }
                        //set ACK PDU
                        memset(&ResponsePDU.data, 0 , sizeof(ResponsePDU.data));
                        ResponsePDU.type = 'A';
                        //send ACK through socket
                        sendto(s, &ResponsePDU, strlen(ResponsePDU.data)+1, 0,(struct
sockaddr *)&fsin, sizeof(fsin));


                        break;

                }
        }
}
```

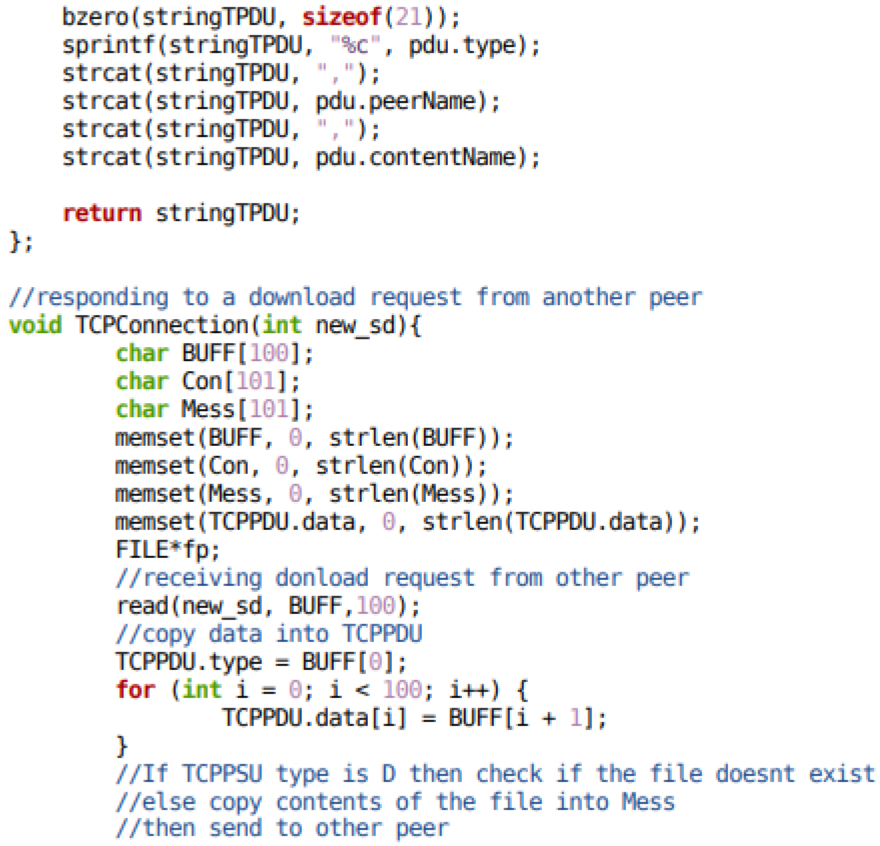
Content Peer Client and Content Peer Server Code



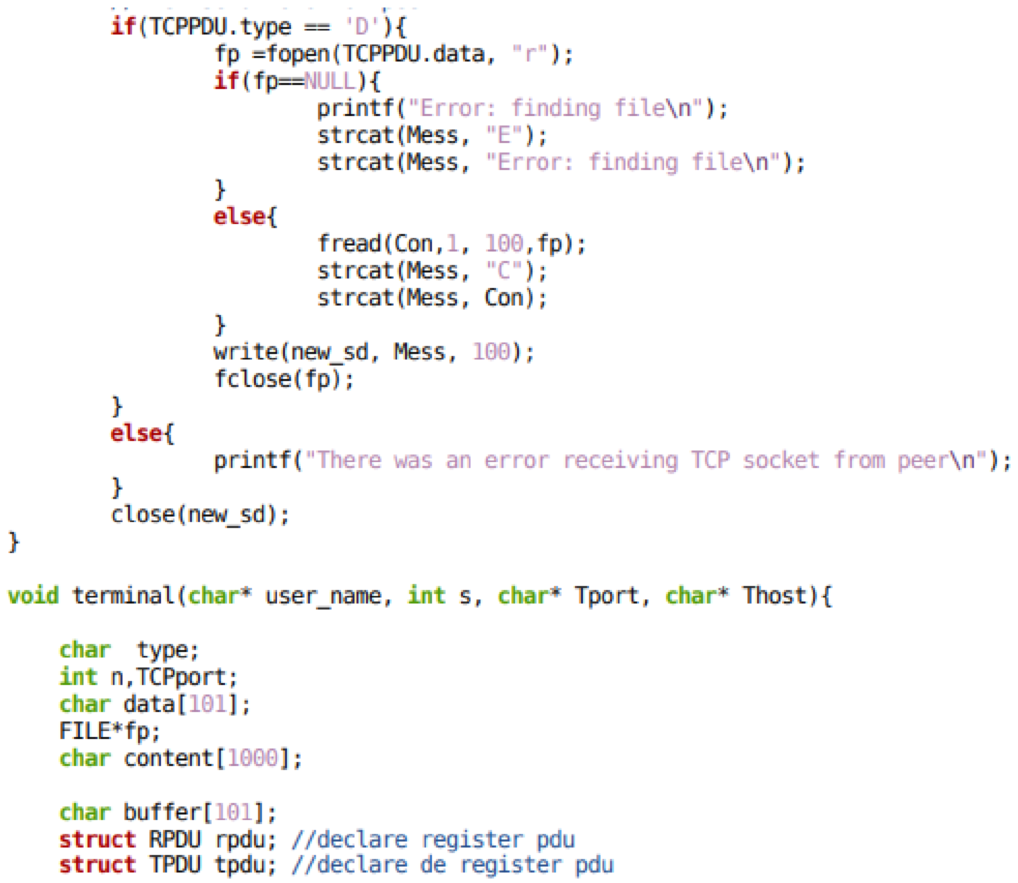
```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <stdio.h>
#include <time.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <unistd.h>


#define BUFSIZE 100
#define ACK_PDU_TYPE 'A'
#define ERROR_PDU_TYPE 'E'

//PDU structure
struct pdu {
char type;
char data[100];
} NamePDU, IndexPDU, SendPDU,TCPPDU;

//Register PDU structure
struct RPDU {
        char type;
        char peerName[10];
        char contentName[10];
        char address[80];
};

//De Register PDU structure
struct TPDU {
        char type;
        char peerName[10];
        char contentName[10];
};
```



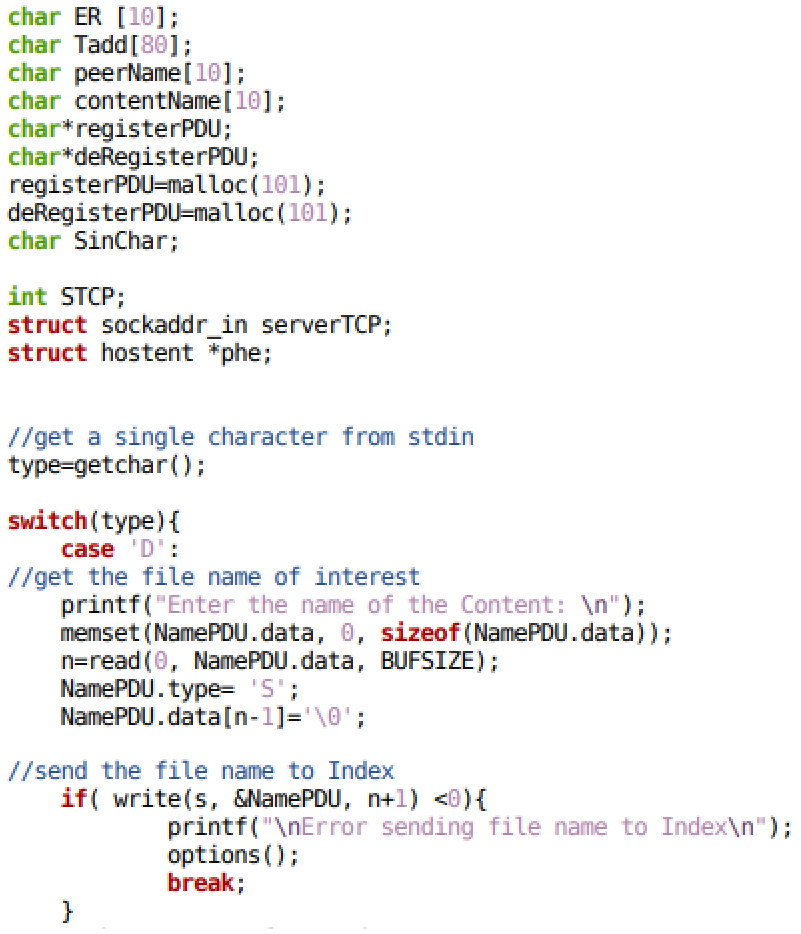
```c
//print all the menu options
void options() {
        printf("\nChoose one of the following options:\nR - Register Content\nD - Content
Download Request\nT - Content De-Registration\nO - List of Online Registered Content\nS -
Search for Content\nQ - Quit\n\n");
}

//convert RPDU to string for transmission
char * genRpduString(struct RPDU pdu, char * stringRPDU){
    bzero(stringRPDU, sizeof(101));
    sprintf(stringRPDU, "%c", pdu.type);
    strcat(stringRPDU, ",");
    strcat(stringRPDU, pdu.peerName);
    strcat(stringRPDU, ",");
    strcat(stringRPDU, pdu.contentName);
    strcat(stringRPDU, ",");
    strcat(stringRPDU, pdu.address);

    return stringRPDU;
};

//convert TPDU to string for transmission
char * genTpduString(struct TPDU pdu, char * stringTPDU){
```



```c
    bzero(stringTPDU, sizeof(21));
    sprintf(stringTPDU, "%c", pdu.type);
    strcat(stringTPDU, ",");
    strcat(stringTPDU, pdu.peerName);
    strcat(stringTPDU, ",");
    strcat(stringTPDU, pdu.contentName);

    return stringTPDU;
};

//responding to a download request from another peer
void TCPConnection(int new_sd){
        char BUFF[100];
        char Con[101];
        char Mess[101];
        memset(BUFF, 0, strlen(BUFF));
        memset(Con, 0, strlen(Con));
        memset(Mess, 0, strlen(Mess));
        memset(TCPPDU.data, 0, strlen(TCPPDU.data));
        FILE*fp;
        //receiving donload request from other peer
        read(new_sd, BUFF,100);
        //copy data into TCPPDU
        TCPPDU.type = BUFF[0];
        for (int i = 0; i < 100; i++) {
                TCPPDU.data[i] = BUFF[i + 1];
        }
        //If TCPPSU type is D then check if the file doesnt exist
        //else copy contents of the file into Mess
        //then send to other peer
```



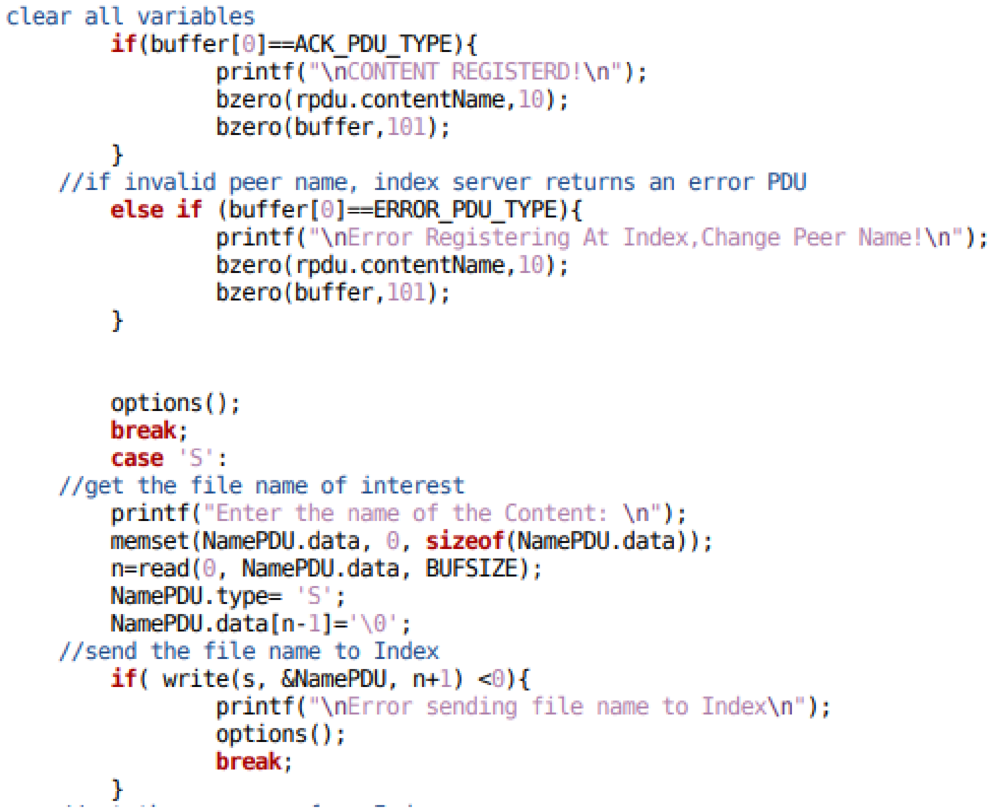
```
        if(TCPPDU.type == 'D'){
                fp =fopen(TCPPDU.data, "r");
                if(fp==NULL){
                        printf("Error: finding file\n");
                        strcat(Mess, "E");
                        strcat(Mess, "Error: finding file\n");
                }
                else{
                        fread(Con,1, 100,fp);
                        strcat(Mess, "C");
                        strcat(Mess, Con);
                }
                write(new_sd, Mess, 100);
                fclose(fp);
        }
        else{
                printf("There was an error receiving TCP socket from peer\n");
        }
        close(new_sd);
}

void terminal(char* user_name, int s, char* Tport, char* Thost){

    char  type;
    int n,TCPport;
    char data[101];
    FILE*fp;
    char content[1000];

    char buffer[101];
    struct RPDU rpdu; //declare register pdu
    struct TPDU tpdu; //declare de register pdu
```



```
char ER [10];
char Tadd[80];
char peerName[10];
char contentName[10];
char*registerPDU;
char*deRegisterPDU;
registerPDU=malloc(101);
deRegisterPDU=malloc(101);
char SinChar;

int STCP;
struct sockaddr_in serverTCP;
struct hostent *phe;


//get a single character from stdin
type=getchar();

switch(type){
    case 'D':
//get the file name of interest
    printf("Enter the name of the Content: \n");
    memset(NamePDU.data, 0, sizeof(NamePDU.data));
    n=read(0, NamePDU.data, BUFSIZE);
    NamePDU.type= 'S';
    NamePDU.data[n-1]='\0';

//send the file name to Index
    if( write(s, &NamePDU, n+1) <0){
            printf("\nError sending file name to Index\n");
            options();
            break;
    }
```

```
//get the response from Index
    memset(data, 0, sizeof(data));
    n=read(s, data, BUFSIZE);
    IndexPDU.type = data[0];
    memset(IndexPDU.data, 0, sizeof(IndexPDU.data));
    for(int i=0; i<n; i++){
            IndexPDU.data[i] = data[i+1];
    }
    IndexPDU.data[n-1] ='\0';
    TCPport = atoi(IndexPDU.data);
//check if there was an error
    if(IndexPDU.type == 'E'){
            printf("Error Content Doesn't Exist\n");
            options();
            break;
    }
//Set up TCP connection with the Peer
//empty out the Server structure and set default Values
memset(&serverTCP, 0 , sizeof(serverTCP));
serverTCP.sin_family = AF_INET;
serverTCP.sin_port = TCPport;

// Make sure everything good with IP and copy into server
if  (phe = gethostbyname(Thost)) {
    memcpy(&serverTCP.sin_addr, phe->h_addr, phe->h_length);
} else if ((serverTCP.sin_addr.s_addr = inet_addr(Thost)) == INADDR_NONE) {
    fprintf(stderr, "TCP:Cant get host entry \n");





}

// Create a stream socket
if ((STCP= socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    fprintf(stderr, "TCP:Can't create socket\n");
}


    printf("Connected to Port: %d\n", serverTCP.sin_port );
// Connect the socket
if (connect(STCP, (struct sockaddr *)&serverTCP, sizeof(serverTCP)) < 0)
    fprintf(stderr, "TCP:Can't connect to %s \n", Thost);

//sending the file name to the peer that has the content
NamePDU.type='D';
write(STCP, &NamePDU, sizeof(NamePDU.data) + 1);
//recieving the reponse from the peer
memset(data, 0, sizeof(data));
bzero(content,1000);
while( n=read(STCP, data, BUFSIZE) ){
        SendPDU.type= data[0];
        for(int i=0; i<n; i++){
            SendPDU.data[i] = data[i+1];
        }
        SendPDU.data[n-1]='\0';
        strcat(content, SendPDU.data);
        memset(SendPDU.data, 0, sizeof(SendPDU.data));
}
```

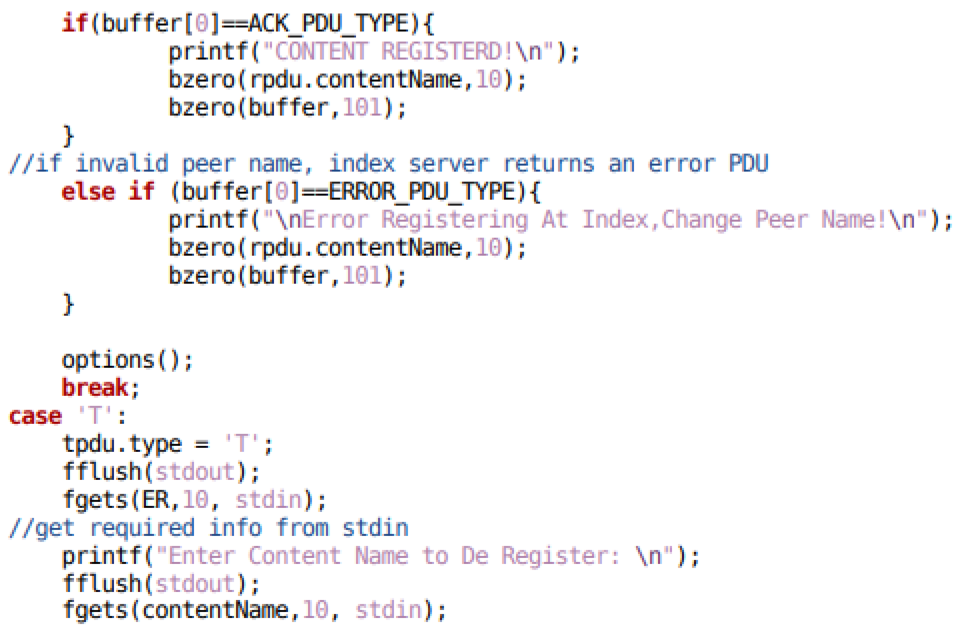
```
//check if there was an error
if(SendPDU.type == 'E'){
    printf("Error receiving from peer");
    options();
    break;
}
//else create file and store contents inside
else{
    fp=fopen(NamePDU.data,"w");
    fputs( content,fp);
    fclose(fp);
    memset(content, 0, sizeof(content));
    printf("%s DOWNLOADED!",NamePDU.data);

}
close(STCP);


    rpdu.type = 'R';
//save info to rpdu
    strcpy(rpdu.peerName,user_name);
    //issue lies here
    strcpy(rpdu.contentName,NamePDU.data);
    sprintf(Tadd, "%d",serverTCP.sin_port);
    strcpy(rpdu.address,Tport);
//convert Rpdu to string
    registerPDU = genRpduString(rpdu, registerPDU);
//write the register PDU string to index server
    write(s,registerPDU,101);
//get response from index
    int receive=0;
    receive = read(s, buffer,101);
//if sucessfully registered, index server returns an Ack PDU, so we let the user know and
```



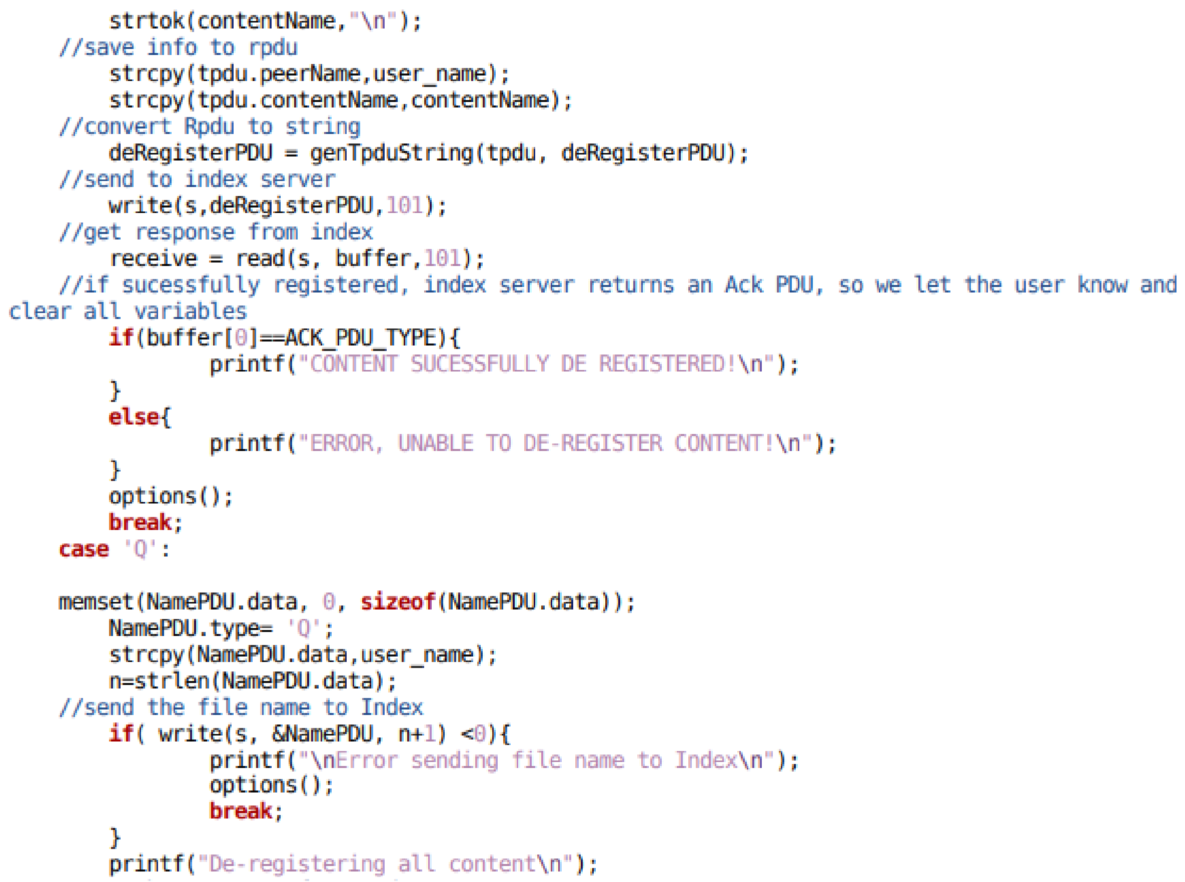
```
clear all variables
        if(buffer[0]==ACK_PDU_TYPE){
                printf("\nCONTENT REGISTERD!\n");
                bzero(rpdu.contentName,10);
                bzero(buffer,101);
        }
    //if invalid peer name, index server returns an error PDU
        else if (buffer[0]==ERROR_PDU_TYPE){
                printf("\nError Registering At Index,Change Peer Name!\n");
                bzero(rpdu.contentName,10);
                bzero(buffer,101);
        }


        options();
        break;
        case 'S':
    //get the file name of interest
        printf("Enter the name of the Content: \n");
        memset(NamePDU.data, 0, sizeof(NamePDU.data));
        n=read(0, NamePDU.data, BUFSIZE);
        NamePDU.type= 'S';
        NamePDU.data[n-1]='\0';
    //send the file name to Index
        if( write(s, &NamePDU, n+1) <0){
                printf("\nError sending file name to Index\n");
                options();
                break;
        }
```

```
//get the response from Index
    memset(data, 0, sizeof(data));
    n=read(s, data, BUFSIZE);
    IndexPDU.type = data[0];
    memset(IndexPDU.data, 0, sizeof(IndexPDU.data));
    for(int i=0; i<n; i++){
            IndexPDU.data[i] = data[i+1];
    }
    IndexPDU.data[n-1] ='\0';
    TCPport = atoi(IndexPDU.data);
//check if there was an error
    if(IndexPDU.type == 'E'){
            printf("Error Content Doesn't Exist\n");
            options();
            break;
    }
//If the server has the content
    else if( IndexPDU.type = 'S'){
            printf("\nContent Can Be Found At Port: %d\n",TCPport);
    }
    options();
    break;
case 'O':

    SinChar='O';
//send the file name to Index
    if( write(s,&SinChar, 1) <0){
            printf("\nError sending file name to Index\n");
            options();
            break;
    }
//get the response from Index
    memset(data, 0, sizeof(data));
```

```
        n=read(s, data, BUFSIZE);
        IndexPDU.type = data[0];
        memset(IndexPDU.data, 0, sizeof(IndexPDU.data));
        for(int i=0; i<n; i++){
                IndexPDU.data[i] = data[i+1];
        }
        IndexPDU.data[n-1] ='\0';
    //check if there was an error
        if(IndexPDU.type == 'E'){
                printf("\nError receiving from Index\n");
                options();
                break;
        }
    //If the server has the content
        else if( IndexPDU.type = 'O'){
                printf("Online Content:\n%s\n",IndexPDU.data);
        }
        options();
        break;
    case 'R':
        rpdu.type = 'R';
        fflush(stdout);
        fgets(ER,10, stdin);
    //get required info from stdin
        printf("Enter Content Name: \n");
        fflush(stdout);
        fgets(contentName,10, stdin);
        strtok(contentName,"\n");
    //save info to rpdu
        strcpy(rpdu.peerName,user_name);
        strcpy(rpdu.contentName,contentName);
        strcpy(rpdu.address,Tport);
    //convert Rpdu to string
        registerPDU = genRpduString(rpdu, registerPDU);
    //write the register PDU string to index server
        write(s,registerPDU,101);
    //get response from index
        receive = read(s, buffer,101);
    //if sucessfully registered, index server returns an Ack PDU, so we let the user know and
clear all variables
```



```
    if(buffer[0]==ACK_PDU_TYPE){
            printf("CONTENT REGISTERD!\n");
            bzero(rpdu.contentName,10);
            bzero(buffer,101);
    }
//if invalid peer name, index server returns an error PDU
    else if (buffer[0]==ERROR_PDU_TYPE){
            printf("\nError Registering At Index,Change Peer Name!\n");
            bzero(rpdu.contentName,10);
            bzero(buffer,101);
    }

    options();
    break;
case 'T':
    tpdu.type = 'T';
    fflush(stdout);
    fgets(ER,10, stdin);
//get required info from stdin
    printf("Enter Content Name to De Register: \n");
    fflush(stdout);
    fgets(contentName,10, stdin);
```



```
      strtok(contentName,"\n");
  //save info to rpdu
      strcpy(tpdu.peerName,user_name);
      strcpy(tpdu.contentName,contentName);
  //convert Rpdu to string
      deRegisterPDU = genTpduString(tpdu, deRegisterPDU);
  //send to index server
      write(s,deRegisterPDU,101);
  //get response from index
      receive = read(s, buffer,101);
  //if sucessfully registered, index server returns an Ack PDU, so we let the user know and
clear all variables
      if(buffer[0]==ACK_PDU_TYPE){
              printf("CONTENT SUCESSFULLY DE REGISTERED!\n");
      }
      else{
              printf("ERROR, UNABLE TO DE-REGISTER CONTENT!\n");
      }
      options();
      break;
  case 'Q':

  memset(NamePDU.data, 0, sizeof(NamePDU.data));
      NamePDU.type= 'Q';
      strcpy(NamePDU.data,user_name);
      n=strlen(NamePDU.data);
  //send the file name to Index
      if( write(s, &NamePDU, n+1) <0){
              printf("\nError sending file name to Index\n");
              options();
              break;
      }
      printf("De-registering all content\n");
```

```
    //get the response from Index
        memset(data, 0, sizeof(data));
        n=read(s, data, BUFSIZE);
        IndexPDU.type = data[0];
        memset(IndexPDU.data, 0, sizeof(IndexPDU.data));
        for(int i=0; i<n; i++){
                IndexPDU.data[i] = data[i+1];
        }
        IndexPDU.data[n-1] ='\0';
        if( IndexPDU.type = 'A'){
                printf("ACKNOWLEDGED!\n");
        }
        exit(0);
  }

}

int main (int argc, char** argv) {

        int     s, new_s, port;
        struct  sockaddr_in server;
        char *host;
        struct hostent *phe;


        int TCPs,alen, new_sd, client_len;
        struct  sockaddr_in TCPserver, client;
        struct hostent *TCPphe;



      int n;
      char user_name[10];
      char ER [10];
      char Tport [80];
      char Thost [80];
      memset(user_name, 0 , sizeof(user_name));
      memset(ER, 0 , sizeof(ER));
      memset(Tport, 0 , sizeof(Tport));
      memset(Thost, 0 , sizeof(Thost));



  //store the IP address and Port # from the terminal
  switch(argc) {
      case 3:
          host = argv[1];
          port = atoi(argv[2]);
          break;
      default:
          fprintf(stderr, "Usage: %s [port]\n", argv[0]);
          exit(1);
  }

  //empty out the Server structure and set default Values
  memset(&server, 0 , sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
```

```
 // Make sure everything good with IP and copy into server
 if  (phe = gethostbyname(host)) {
     memcpy(&server.sin_addr, phe->h_addr, phe->h_length);
 } else if ((server.sin_addr.s_addr = inet_addr(host)) == INADDR_NONE) {
     fprintf(stderr, "Cant get host entry \n");
 }

 // Create a stream socket
 if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
     fprintf(stderr, "Can't create socket\n");
     exit(1);
 }

 // Connect the socket
 if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
     fprintf(stderr, "Can't connect ot %s \n", host);


 //Set up TCP connection
 //empty out the Server structure and set default Values
 memset(&TCPserver, 0 , sizeof(TCPserver));
 TCPserver.sin_family = AF_INET;
 TCPserver.sin_port = htons(0);
 TCPserver.sin_addr.s_addr=htonl(INADDR_ANY);

 // Create a stream socket
 if ((TCPs = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
     fprintf(stderr, "TCP:Can't create socket\n");
 }

 if (bind(TCPs,(struct sockaddr *)&TCPserver, sizeof(TCPserver)) == -1){
             fprintf(stderr, "Can't bind name to socket\n");
             exit(1);



    }
//Assign a port number
    alen=sizeof(struct sockaddr_in);
    getsockname(TCPs,(struct sockaddr *)&TCPserver, &alen);

//listen to any pending connections
    listen(TCPs, 5);
    fd_set rfds, afds;

//Get the user name
    printf("Choose a user name\n");
    memset(user_name, 0, sizeof(user_name));
    n=read(0, user_name, sizeof(user_name));
    user_name[n-1]='\0';
//convert TCP port number into  string
    sprintf(Thost, "%d",TCPserver.sin_addr.s_addr);
    printf("\n%s's TCP port number: %d\n", user_name,TCPserver.sin_port);
    sprintf(Tport, "%d",TCPserver.sin_port);
    options();
```

```
        while(1){
        //clear the file descriptor
                FD_ZERO(&afds);
        //gets TCPS value sets the corresponding afds bit to 1
                FD_SET(TCPs, &afds);
        //sets the 0th bit of afds to 1
                FD_SET(0, &afds);
                memcpy(&rfds, &afds, sizeof(rfds));
        // retrun is blocked until TCP connection estblished
        //or get someting from stdin then rfds will contain
        //the socket that needs to be serviced
                select(FD_SETSIZE, &rfds,NULL,NULL,NULL);

        //check if there is a pending TCP connection
        //Then if unable to accept the TCP connection print error
        //else run TCPconnection function
                if(FD_ISSET(TCPs,&rfds)){
                        client_len = sizeof(client);
                        new_sd = accept(TCPs, (struct sockaddr *)&client, &client_len);
                        if(new_sd < 0){
                            fprintf(stderr, "TCP Can't accept client \n");
                        }
                        else{
                                TCPConnection(new_sd);
                        }

                }

        //check if data arrived at stdin and run terminal function
                if(FD_ISSET(0,&rfds)){
                        terminal(user_name,s,Tport,Thost);
                }
        }
}
```