In this application, you require implementing three C programs, namely Client, Proxy Server (which will act both as client and server) and DNS Server, and they communicate with each other based on TCP sockets. The aim is to implement a simple 2 stage DNS Resolver System.

1) Initially, the client will connect to the proxy server using the server's TCP port already known to the client.
2) After successful connection, the client sends a Request Message (Type 1/Type 2) to the proxy server.
3) The proxy server has a limited cache (assume a cache with three IP to Domain_Name mapping entries only).
4) After receiving the Request Message, proxy server based on the Request Type (Type 1/Type 2) searches its cache for corresponding match. If match is successful, it will send the response to the client using a Response Message. Otherwise, the proxy server will connect to the DNS Server using a TCP port already known to the Proxy server and send a Request Message (same as the client).
5) The DNS server has a database (say .txt file) with it containing set of Domain_name to IP_Address mappings. Once the DNS Server receives the Request Message from proxy server, it searches in its file for possible match and sends a Response Message (Type 3/Type 4) to the proxy server.
6) On receiving the Response Message from DNS Server, the proxy server forwards the response back to the client.
7) If the Response Message type is 3, then the proxy server must update its cache with the fresh information using FIFO scheme.
8) After each negotiation phase, the TCP connection on both sides should be closed gracefully releasing the socket resource.

To run dns client-> ./dncClient 127.0.1.5 8090
Where 127.0.1.5 is proxy server ip address and 8090 is proxy server port number.
/proxyserver 8090
/dnsserver 8080

- Socket() – Endpoint for communication
- Bind() - Assign a unique telephone number.
- Listen() – Wait for a caller.
- Connect() - Dial a number.
- Accept() – Receive a call.
- Send(), Recv() – Talk.
- Close() – Hang up.

Steps for code for dnsserver:

1) Take the input msg in char array with buffer size large enough.
2) The socketaddr_in is a struct with sin_family, ip addr and port number. So we create sockadr_in for both server and client.
3) Now check the command line arguments if not 2 in case of dns server return invalid and exit(EXIT_FALIURE)

4) The second argument is port number of dns server so convert it into integer and store it. If not valid port number return invalid (0-65535 valid).
5) socket(AF_INET, SOCK_STREAM, 0); This call results in **a stream socket with the TCP protocol providing the underlying communication**. AF_INET is **an address family that is used to designate the type of addresses that your socket can communicate with** (in this case, Internet Protocol v4 addresses). When you create a socket, you have to specify its address family, and then you can only use addresses of that type with the socket. SOCK_STREAM. **Provides sequenced, two-way byte streams with a transmission mechanism for stream data**. This socket type transmits data on a reliable basis, in order, and with out-of-band capabilities.
6) SOCK_DGRAM is a datagram-oriented socket, regardless of the transport protocol used. UDP is one, but not the only, transport that uses datagrams. SOCK_STREAM is a stream-oriented socket, regardless of the transport protocol used. TCP is one, but not the only, transport that uses streams.
7) If the value of socket(AF_INET, SOCK_STREAM, 0) is less than 0 then it means socket creation failed exit.
8) Set the sin_family as AF_INET and port number as the one received in input but convert it using htons(The htons function **takes a 16-bit number in host byte order and returns a 16-bit number in network byte order used in TCP/IP networks basically converting it to little endian (LSB first)).**
9) Assign the IP address of dns server to socketaddr_in.
10) Now bind serversocketid and servaddr(i.e. socketaddr_in) and if it returns < 0 return error of failure.
11) Listen serversocketid and if it returns < 0 return error of failure.
12) The inet_addr function **converts a string containing an IPv4 dotted-decimal address into a proper address for the IN_ADDR structure** wherease inet_ntoa() accepts an Internet address expressed as a 32-bit quantity in network byte order and returns a string expressed in dotted-decimal notation.
13) Accept serversocketid and servaddr and if it returns < 0 return error of failure.
14) Send greeting message stored in through clntsocketid with length of msg also as parameter. The function returns the number of bytes send if it is not eq to input msg return failure msg.
15) Now comes HandleTCPclient(clntsocketid) function. (explained below) after this function the serversocketid is closed if no req comes.

Handle TCP client:

1) Create arrays for receiving and sending msgs.
2) Create two maps for dn->ip and ip->dn.
3) Open the input file of dns database and fill the maps and close it.
4) The getpeername function **retrieves the address of the peer connected to the socket s and stores the address in the SOCKADDR structure identified by the name parameter**. This function works with any address

family and it simply returns the address to which the socket is connected. So now we have address in clntaddr of clntsktid.

5) recv(clntSktId, DNSRecvBuffer, RCVBUFSIZE, 0) if <0 you know.
6) If this is type 1 (found out using first char and rest is dn) then extract dn and find it in map. If found add 3(IPaddr) in sender buffer else 4notfound.
7) If type 2 similar for other map.
8) If 0 then close the connection ->close(clntsktid).
9) If not 0 then send the sendbuffer using send(clntSktId, DNSSendBuffer, strlen(DNSSendBuffer), 0)

bind() is typically used on the server side, and associates a socket with a socket address structure, i.e. a specified local IP address and a port number. listen() is used on the server side, and causes a bound TCP socket to enter listening state.

Steps for DNSclient:

1) Create structure of type sockarr_in for server and client as servaddr and clnt addr.
2) If argument from command line are not 3 (filename server-ipaddr server-port) return failure.
3) clntsktid=Create socket(AF_NET,SOCK_STREAM,0)
4) Assign sin_family as AF_NET, sin_port as htons(servport), sin_addr as inet_addr(arg[1]) in serv_addr.
5) connect (clntsktid,servaddr)<0
6) You can print welcome msg, and now receive msg using clntsktid in receiver buffer using command -> recv(clntsktid,wlcmMsg,sizeofbuffer,0)<0.
7) HandleTCPserver(clntsktid) after this close the cntsktid.

HandleTCPserver:

1) Create receiver and sender buffer
2) Show the list of choices available -> type 1 dn->ip , type 2 ip->dn and 0 to close connection. If other than this print invalid and ask again.
3) Take the msg in recv buffer (geline function) with type concatenated in the prefix.
4) Send this msg to server using command send(clntsktid,clientmsgbuff, len,0)<0.
5) Receive the output (dn/ip) using recv(clntsktid,clientrecbuff,sizeofbuff,0)<0. If msg type is 0 then exit else print msg. Else print input and output string.
6) Now ask to continue (y/n) and repeat accordingly.

Steps for proxy server:

1) Create structure of type sockarr_in for server and client sy servaddr and clnt addr.
2) Check args number is 2 or not (proxy server port and filename) also check validity of port number entered.
3) Create servsktid=socket(AF_NET,SOCK_STREAM,0)
4) Assign sin_family as AF_NET, sin_port as htons(servport), sin_addr as inet_addr(proxy server ip known already) in serv_addr.
5) bind(servsktid,servaddr,size)<0
6) listen(servsktid,maxpending)<0
7) clntsktid=accept(servsktid,clntaddr,clntlen)<0

8) Send the information of connection established using send(clntsktid,proxyshortmsg,buffsize,0)<0.
9) Open proxy cache file, put everything in list name cache (list of pair of strings).
10) The getpeername function **retrieves the address of the peer connected to the socket s and stores the address in the SOCKADDR structure identified by the name parameter**. This function works with any address family and it simply returns the address to which the socket is connected. So now we have address in clntaddr of sd.
11) recv(sd,recvbuff,sizeofbuff,0)<0 and convert it to string.
12) If the first char is 1 then find the string in first of pair and if found add 3(result) in sendbuff. If first char is 2 then search in second of pair and add 3(result) in sendbuff. If not found 4notfound.
13) If there was miss in cache call proxy_to_dnsserver (explained below)
14) Send msg to client using sent(sd,sendbuff,sizeofbuff,0)<0.

Proxy_to_dnsserver code:

1) Create sockaddr_in for dnsservaddr.
2) Create clntsktid=socket(AF_NET,SOCK_STREAM,0)<0.
3) Assign sin_family as AF_NET, sin_port as htons(dns servport), sin_addr as inet_addr(dns server ip) in serv_addr.
4) connect (clntsktid,dnsservaddr,size)<0
5) send (clntsktid,recvbuff,sizeofbuff,0)<0 (send query to dns server)
6) recv(clntsktid,sendbuff,sizeofbuff,0)<0
7) Convert this recvbuff (1www.google.com) and sendbuff(31.1.1.1)
8) If send buff starts with 3 then if cache size is less than 3 add the mapping to cache depending on typ1/2.
9) If cache size is 3 , pop front and add this mapping to last. And update cache using the this list.
10) Close the file and the clntsktid.