```c
#include<netdb.h>
#include<stdlib.h>
#include <unistd.h>
#include<errno.h>
#include<stdio.h>
#include<string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h> /* superset of previous */

struct hostent * he;  //Puntatore alla struttura IPv4 dell'hostname

int lunghezza; //Lunghezza indirizzo per accept()
int yes=1; //Flag utilizzato per settare un opzione del socket

//Struttura singono header (nome, valore)
struct header {
  char * n;
  char * v;
};
struct header h[100];//Header (array di strutture)

// Indirizzo IPv4
struct sockaddr_in indirizzo,indirizzo_server;
struct sockaddr_in indirizzo_remoto;

int primiduepunti;  // Flag usato per leggere gli headers

char request[10000];
char request[10000];
char proxyrequest[10000];
char response[10000];
char* request_line; //Puntatore alla request line
char * method, *uri, *http_ver; //Puntatori che indicano i parametri della RequestLine
char * scheme, *hostname, *filename; //Puntatori che indicano i parametri della URI

int c;  //Carattere utilizato per leggere ogni carattere del file
FILE *fin;  //File che dovrà essere trasmesso

int main(){
  int s,s2,s3,t,j,i; //s:socket per ricevere, t:variabile di supporto per gli errori
  char command[1000]; //Stringa che contiene il comando di shell eseguire
  char car; //Singolo carattere ricevuto dal server

//APRIRE COMUNICAZIONE
  /*socket restituisce un INT che è un file descriptor
    ovvero l'indice della tabella con tutto ciò che serve per gestire la comunicazione*/
  s = socket(AF_INET,SOCK_STREAM,0);
  if (s == -1){
    perror("Socket Fallita");
    return 1;
  }
  /* Opzioni del socket: opzioni a levello socket(SOL_SOCKET),
                         riutilizzo degli indirizi(SO_REUSEADDR),
                         1(si),
                         sizeof(int)
     ritorna 0 se tutto va bene, altrimenti -1*/
```

```c
  if ( setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1 ) {
    perror("setsockopt");
    return 1;
  }

//CONNESSIONE
  indirizzo.sin_family=AF_INET; //IPv4
  indirizzo.sin_port=htons(7888); //Porta
  indirizzo.sin_addr.s_addr=0;

  // Bind assegna un indirizzo al socket, t=0 se tutto va bene altrimenti -1
  t=bind(s,(struct sockaddr *) &indirizzo, sizeof(struct sockaddr_in));
  if (t==-1){
    perror("Bind fallita");
    return 1;
  }

  //Listen: marca il socket s come passivo, ovvero che deve ricevere
  t=listen(s,10);
  if(t==-1){
    perror("Listen Fallita");
    return 1;
  }

  while( 1 ){
    lunghezza = sizeof(struct sockaddr_in);

    //Accept: restituisce il file descriptor di un nuovo socket che
    // eredita tutto da quello vecchio ma è già connesso,
    // pronto per il read e write. Questo perchè l'altro è impegnato con listen.
    s2=accept(s,(struct sockaddr *)&indirizzo_remoto, &lunghezza);

    //Fork() crea un processo figlio che eredita tutto dal padre e differisce solo per il PID
    //restituisce 0 se va tutto bene altrimenti -1
    //Usata perchè così una connessione da un client è già accettata
    //e possono essere eseguiti più processi senza eseguirli uno per uno
    if (fork()==0){
      if (s2 == -1){
        perror("Accept Fallita");
        return 1;
      }

      //RICEVO RICHIESTA
      /*Full-Request = Request-Line
                      *( General-Header
                       | Request-Header
                       | Entity-Header )
                       CRLF
                       [ Entity-Body ]   */
      // Request-Line = Method SP Request-URI SP HTTP-Version CRLF   (con CRLF=\r\n)
      // Header = nome: valore

      //Requst Line:
      h[0].n=request;
      request_line=h[0].n;
      h[0].v=h[0].n;
```

```
//Headers:
for(i=0,j=0; (t=read(s2,request+i,1))>0;i++){
  if (( i>1) && (request[i]=='\n') && (request[i-1]=='\r')){
    primiduepunti=1;
    request[i-1]=0;
    if(h[j].n[0]==0) break;
    h[++j].n=request+i+1;
  }
  if (j>0 && primiduepunti && (request[i]==':')){
    h[j].v = request+i+1;
    request[i]=0;
    primiduepunti=0;
  }
}
if (t == -1 ) {
  perror("Read Fallita");
  return 1;
}

// Visualizzo a schermo la RequestLine e gli Headers ricevuti
printf("Request: %s\n",request_line);
for(i=1;i<j;i++){
  printf("%s ===> %s\n",h[i].n,h[i].v);
}

// Ricavo Method, URI e HTTP-Version dalla RequestLine
method = request_line;
for(i=0;request_line[i]!=' ' && request_line[i];i++){};
if (request_line[i]!=0) { request_line[i]=0; i++;}
uri = request_line + i;
for(;request_line[i]!=' ' && request_line[i];i++){};
if (request_line[i]!=0) { request_line[i]=0; i++;}
http_ver = request_line +i;

//Visualizzo a schermo Method, URI e HTTP-Version
printf("Method = %s, URI = %s, Http-Version = %s\n", method, uri, http_ver);


//Se il Method non è GET rispondo che non è implementato
if (strcmp(method,"GET")){//se methon=GET restituisce 0
  sprintf(response,"HTTP/1.1 501 Not implemented\r\n\r\n");
  write(s2,response,strlen(response));
  close(s2);
  continue; // esco dal if(fork()==0)
}

//Ricavo lo scheme (http) dall URI
scheme = uri;
for(i=0;uri[i]!=':' && uri[i];i++);
if(uri[i]!=0){uri[i]=0; i++;}
if((uri[i]=='/') && (uri[i+1]=='/')) i=i+2;//http://
else {
  printf("Errore hostname%s",uri+i);
  close(s2);
  continue;
}
```

```
      //Ricavo hostname dall URI
      hostname = uri+i;
      for(;uri[i]!='/' && uri[i];i++);
      if(uri[i]!=0){uri[i]=0; i++;}

      //Ricavo il filename dall URI
      filename = uri+i;

      //Visualizzo a schermo scheme, hostname e filename
      printf("Scheme:%s, hostname:%s, filename:%s\n",scheme,hostname,filename);

      // Ritorna una struct IPv4 dato l'hostname
      he = gethostbyname(hostname);
      printf("Indirizzo di %s : %d.%d.%d.%d\n",hostname, (unsigned char)  (he->h_addr[0]),
      (unsigned char) (he->h_addr[1]), (unsigned char) (he->h_addr[2]), (unsigned char)
      (he->h_addr[3]));


    //COMUNICAZIONE COL SERVER
      //APRIRE COMUNICAZIONE
      /*socket restituisce un INT che è un file descriptor
        ovvero l'indice della tabella con tutto ciò che serve per gestire la comunicazione*/
      s3 = socket(AF_INET,SOCK_STREAM,0);
      if (s == -1){
        perror("Socket Fallita");
      }

      //CONNESSIONE
      indirizzo_server.sin_family  = AF_INET; //IPv4
      indirizzo_server.sin_port = htons(80);  //Porta
      indirizzo_server.sin_addr.s_addr = *((uint32_t *)he->h_addr);

      t=connect(s3,(struct sockaddr *)&indirizzo_server, sizeof(struct sockaddr_in));
      if ( t== -1){
        perror("Connect fallita\n");
      }

      //RICHIESTA
      // Preparo la richiesta: (scrivo sull'array di char proxyrequest)
      sprintf(proxyrequest,"GET /%s
      HTTP/1.1\r\nHost:%s\r\nConnection:Close\r\n\r\n",filename,hostname);
      printf("proxyrequest: %s",proxyrequest);

      // Invio la richiesta:
      write(s3,proxyrequest,strlen(proxyrequest));

      //Leggo la risposta dal Server
      while(read(s3,&car,1)){
        write(s2,&car,1); //Scrivo al Client quello che leggo dal server
        //printf("%c",car);
      }
      close(s2);
      close(s3);
      exit(0);
    }//Fine fork()
  }//Fine while(1)
}//Fine main()
```