```c
#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 */

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;
struct sockaddr_in indirizzo_remoto;

int primiduepunti;  // Flag usato per leggere gli headers
char request[10000];
char response[10000];
char * request_line;  //Puntatore alla request line
char * method, *uri, *http_ver; //Puntatori che indicano i parametri della RequestLine

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

int main(){
  int s,s2,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

//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); //AF_INET=IPv4, SOCK_STREAM:apre uno stream, protocol:0
  (default)
  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*/
  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);
    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 (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 Line: %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 l'URI è "/exec/"
    if(!strncmp(uri,"/exec/",6)){//confronto i primi 6 caratteri
      sprintf(command,"%s > filetemp\n",uri+6);//Metto in "filetemp" il file richiesto
      dall'URI
      printf("Eseguo il comando %s",command);
      if((t=system(command))==0){//Eseguo il comando
        uri = "/filetemp";
      }
      else printf("system ha restituito %d\n",t);
    }

    if ((fin = fopen(uri+1,"rt"))==NULL){// Se non riesco ad aprire il file invio "FIle non
    trovato"
      printf("File %s non aperto\n",uri+1);
      sprintf(response,"HTTP/1.1 404 File not found\r\n\r\n<html>File non trovato</html>");
      t=write(s2,response,strlen(response));
      if (t==-1){
        perror("write fallita");
        return -1;
      }
    } else {//Altrimenti invio il file
      sprintf(response,"HTTP/1.1 200 OK\r\n\r\n");
      t=write(s2,response,strlen(response));
      while((c = fgetc(fin))!=EOF){//fgetc ritorna il carattere sucessivo
        if (write(s2,(unsigned char *)&c,1)!=1){
          perror("Write fallita");
        }
      }
      fclose(fin);
    }
    close(s2);  //Chiudo il socket che risponde
  }//Fine while
}//Fine main
```