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

// Prototipi di funzioni
unsigned short myhtons(unsigned short s);
unsigned char* myhtonx(unsigned char * s,unsigned char * o, int size);
long hexToLong(char * str);

// Indirizzo IPv4
struct sockaddr_in indirizzo;

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

int main(){
  long dim;
  int s,k,t,i,j; //s:file descriptor socket per inviare
  char *p;
  char primiduepunti, chunked=0; //chunked indica se l'header Transfer-Encodind=chunked
  char *status_line, *entity, *chunksize; // Puntatori a: StatusLine, EntityBody, ChunkSize
  int content_length; //Valore che verrà reperito dall'header Content-Length
  char request[1000];
  char response[100000];
  unsigned short u,v;
  u=0xABCD;
  myhtonx((unsigned char*) &u,(unsigned char*) &v,sizeof(unsigned short));
  printf("%X %X\n",u,v);

//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");
  }

//CONNESSIONE
  indirizzo.sin_family  = AF_INET;  //IPv4
  indirizzo.sin_port = htons(80); //Porta80
  ((unsigned char *)&(indirizzo.sin_addr.s_addr))[0] = 147;
  ((unsigned char *)&(indirizzo.sin_addr.s_addr))[1] = 162;
  ((unsigned char *)&(indirizzo.sin_addr.s_addr))[2] = 235;
  ((unsigned char *)&(indirizzo.sin_addr.s_addr))[3] = 155;
  // google.com 216.58.213.228
  // google.co.uk 74.125.206.94
  // unipd.it 147.162.235.155
```

```
  t=connect(s,(struct sockaddr *)&indirizzo, sizeof(struct sockaddr_in));
  if (t== -1) perror("Connect fallita\n");

//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

  // Preparo la richiesta: (scrivo sull'array di char request)
  sprintf(request,"GET /?gfe_rd=cr&dcr=0&ei=7qmvWvjcMdfEaM21i4AK
  HTTP/1.1\r\nHost:www.google.co.uk\r\n\r\n");
  //sprintf(request,"GET / HTTP/1.1\r\nHost:www.google.com\r\n\r\n");

  // Invio la richiesta:
  write(s,request,strlen(request)); // Scrivo sul file descriptor s del socket


//RICEVO LA FULL-RESPONSE
  /*Full-Response = Status-Line
                   *( General-Header
                   | Response-Header
                   | Entity-Header )
                   CRLF
                   [ Entity-Body ]   */
  // Status-Line = HTTP-version SP Status-Code SP Reason-Phrase CRLF
  // Header = nome: valore

  // Status Line:
  h[0].n=response;
  status_line=h[0].n;
  h[0].v=h[0].n;

  //Headers
  for(i=0,j=0; read(s,response+i,1); i++){
    if ((i>1) && (response[i]=='\n') && (response[i-1]=='\r')){
      primiduepunti=1;
      response[i-1]=0;
      if(h[j].n[0]==0) break;
      h[++j].n=response+i+1;
      }
    if (primiduepunti && (response[i]==':')){
      h[j].v = response+i+1;
      response[i]=0;
      primiduepunti=0;
    }
  }

  //Visualizzo a schermo La StatusLine e gli Headers ricevuti
  printf("Status Line: %s\n",status_line);
  content_length=0;
  for(i=1;i<j;i++){
```

```
    printf("%s ===> %s\n",h[i].n,h[i].v);
    if(strcmp("Content-Length",h[i].n)==0)
      content_length= atoi(h[i].v); //Atoi:string to integer
    else if(strcmp("Transfer-Encoding", h[i].n)== 0 && strcmp(" chunked", h[i].v) ==0)
    chunked=1;
  }

  //Se la dimensione non è nulla:
  if(content_length!=0){
    entity = malloc(content_length+1);  //alloca la memoria per l'entity body
    //Legge (content_lenghtsocket-i)byte dal socket s partendo dall'indirizzo (entity+i)
    for(i=0;(i<content_length) && (t=read(s,entity+i,content_length-i));i=i+t);
    //Visualizzo a schermo il BODY
    entity[i]=0;
    printf("%s",entity);
    free(entity);
  }
  else if(chunked){
    chunksize=response+i;
      for(k=0; (t=read(s, response +i+k , 1))>0;  k++){
        if(response[i+k]== '\n' && response[i+k-1]=='\r'){
          response[i+k-1]=0;
          dim = hexToLong(chunksize);
          printf("<%s> %ld\n",chunksize,dim);
          for(j=0;(t=read(s, response +i+k, dim-j))>0 && j<dim; j+=t, k+=t);
          t=read(s,response +i+k, 2);
          //response[i+k+t]=0;
          //printf("***%s***\n",response+i+k);

          k+=2;
          if(dim==0){ printf("Fine body"); break;}
          chunksize=response+i+k;
          k--;
        }

      }
  }
  if (t<0){
    perror("read fallita");
    return 1;
  }

  //Visualizzo a schermo il BODY
  //entity[i]=0;
  //printf("%s",entity);
  //free(entity);


  /*
  p = response;
  while (t=read(s,p,100000)){
          p = p + t;
  *p=0;
  }
  */
}//Fine main
```

```c
long hexToLong(char * str){
  long val=0;
  int n,k;
  for(k=0; str[k]; k++){
    if(str[k]<='9' && str[k]>='0') n=str[k]-'0';
    if(str[k]<='Z' && str[k]>='A') n=str[k]-'A'+10;
    if(str[k]<='z' && str[k]>='a') n=str[k]-'a'+10;
    val= val *16 + n ;
  }
  return val;
}

unsigned short myhtons(unsigned short s){
  unsigned short tmp = 1;
  unsigned char * p;
  p = (unsigned char *) &tmp;
  if (p[0]){
    p[0]=((unsigned char*)&s)[1];
    p[1]=((unsigned char*)&s)[0];
  }
  else tmp = s;

  return tmp;
}

unsigned char*  myhtonx(unsigned char * s,unsigned char * o, int size){
  unsigned short tmp = 1;
  int i;
  unsigned char * p;
  unsigned char appoggio;
  p = (unsigned char *) &tmp;
  if(p[0])
    for(i=0;i<size/2;i++){
      appoggio = s[i];
      o[i]=s[size-i-1];
      o[size-i-1]=appoggio;
    }
  return o;
}
```