Codice sorgente del CLIENT

```
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <sys/time.h>
#include <sys/errno.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#include "define.h"

#define SOCKET_ERROR   ((int)-1)
#define X_init 3
#define Y_init 10
#define LARGHEZZA 70
#define ALTEZZA_1 10
#define ALTEZZA_2 12
#define BORDO_TABELLA BG_BLUE
#define COL_SCRITTE YELLOW
#define BOLD     1
#define BLINK    5
#define REVERSE  7
#define NORMAL   0
#define cprintf printf


//================================================== Funzioni di GraFica ================

/* Enumerazione dei colori */
enum fgcolor {BLACK = 30 , RED , GREEN , BROWN , BLUE , MAGENTA , CYAN , WHITE , YELLOW};
enum bgcolor {BG_BLACK = 40 , BG_RED , BG_GREEN , BG_BROWN , BG_BLUE , BG_MAGENTA , BG_CYAN , BG_WHITE};

//-----------------------------------------------------------------------------------------
void gotoxy(int x , int y);
        /* Permette di spostarsi in un certo punto dello schermo definito dalle coordinate
                x ed y
                - x = ascisse
                - y = ordinata */
//-----------------------------------------------------------------------------------------
void clrscr();
        /* Pulisce lo schermo in modo da rendere più gradevole la grafica */
//-----------------------------------------------------------------------------------------
void _textattr(int attrib);
        /* Imposta il testo, sia foreground che background, con un certo
                attributo attrib */
//-----------------------------------------------------------------------------------------
int grafica_iniziale(char *richiesto , char *salvato , int filesize);
        /* Scriviamo su stdout, appena pulito, il titolo, la tabella e altre scritte
                - richiesto = file chiesto dal server al proxy
                - salvato = nome con cui il client salva il file scaricato dal proxy
                - filesize = dimensione del file */
//-----------------------------------------------------------------------------------------
void disegna_asterischi(int filesize , int segmento);
        /* disegna dei caratteri in una tabella e rappresenta l'avanzamento del download del file
                - filesize = dimensione del file
                - segmento = n° del segmento corrente*/
//-----------------------------------------------------------------------------------------
void percentuale (int filesize , int byte);
        /* stampa la percentuale del file ricevuto durante la sua trasmissione
                - filesize = dimensione del file
                - byte = n° del byte corrente */
//-----------------------------------------------------------------------------------------
int velocita (struct timeval *inizio , struct timeval *fine , int segmento);
        /* scrive su stdout la velocità media con cui riesce a scaricare il file; viene aggiornato ogni volta che viene chiamata,
                - inizio = struttura timeval che contiene il tempo iniziale
                - fine = struttura timeval che contiene il tempo finale
                - segmento = n° del segmento corrente */
//-----------------------------------------------------------------------------------------
void rimanenza (int segmento , int filesize , float velocita);
        /* Scrive su stdout quanti secondi *dovrebbero* mancare al termine del trasferimento del file
                - segmento = n° del segmento corrente
                - filesize = dimensione del file
                - velocita = valore di ritorno della funzione velocita(), indica la vel. di trasferimento del file */
//=========================================================================================


//============================================================ Lettura e Scrittura ============
ssize_t writen (int fd, const void *buf, size_t n){
        size_t nleft;
        ssize_t nwritten;
        char *ptr;

        ptr = (char*)buf;
        nleft = n;
        while (nleft > 0) {
```

```
                if ( (nwritten = write(fd , ptr , nleft)) <= 0) {
                        if (errno == EINTR)
                                nwritten = 0;
                        else
                                return -1;
                        }
                nleft -= nwritten;
                ptr += nwritten;
                }
        return n;
        }

//-------------------------------------------------------------------------------------------------------

int readn(int fd, char *ptr, int nbytes, int mode);
/* Usata per lettura "sicura"; finché non riceviamo nbytes cicla, altrimenti restituisce un errore;
        - fd = file descriptor da cui ricevere i dati
        - ptr = puntatore alla stringa dove andremo a salvare i dati entrati
        - nbytes = n° di byte da leggere
        - mode = se 1 stampa i dati relativi alla ricezione del file sullo schermo (velocità, percentuale, etc)
                altrimenti non stampa nulla */

int readn(int fd, char *ptr, int nbytes, int mode){
        int nleft , nread , ris;
        struct timeval end ,inizio;

        nleft = nbytes;
        gettimeofday(&inizio , NULL);
        while(nleft > 0){
                nread = read(fd , ptr , 1);
                if (mode){                                              /* parte grafica */
                        disegna_asterischi(nbytes , nbytes-nleft);      /* barra di incremento */
                        percentuale(nbytes , nbytes-nleft + 1);         /* percentuale ricevuta */
                        gettimeofday(&end , NULL);                      /* per il calcolo del tempo impiegato */
                        ris = velocita(&inizio , &end , nbytes-nleft);  /* velocità di trasmissione dati */
                        rimanenza(nbytes , nbytes-nleft , ris);         /* tempo stimato rimanente */
                        }
                if(nread<0)
                        return(-1);
                else
                        if(nread == 0) {
                                return(0);
                                break;
                                }
                nleft -= nread;
                ptr   += nread;
                }
        return(nbytes);
        }
//=====================================================================================================




//-------------------------------------------------------------------------------------------
void usage(char *name);
/* stampa un messaggio di errore su stdout in caso di errato numero di parametri passati al programma */

void usage(char *name){
        printf ("Error: invalid number of args \n");
        printf ("Usage: %s server_address server_port file output_file\n" , &name);
        fflush (stdout);
        }
//-------------------------------------------------------------------------------------------
GetRequestHeader makerequest (char * filename);
/* Crea un pacchetto di tipo GetRangeRequest da spedire al server per richiedere un file
        - filename = nome del file richiesto */

GetRequestHeader makerequest (char * filename){
        GetRequestHeader request;
        memset(&request , 0 , sizeof(request));
        request.request_code = 'G';                 /* tipo del pacchetto: G [GET] */
        if (strlen(filename) > FILENAMELEN - 1){
                _textattr(RED);                     /* controllo dell'errore */
                _textattr(BOLD);
                printf("Errore: Nome file troppo grande\n");
                _textattr(NORMAL);
                exit(1);
                }
        else
                strcpy(request.filename , filename);  /* se tutto ok copio il nome del file nel pacchetto */
        return request;
        }
//-------------------------------------------------------------------------------------------
void segnale();
/* Handler per gestire l'interrupt generato dall'utente che ha premuto CTRL + C;
        sposta il cursore in basso e poi termina l'esecuzione del programma, per
        non lasciare delle scritte in mezzo allo schermo */

void segnale(){
```

```
        gotoxy(1,29);
        signal(SIGINT , SIG_DFL);
        signal(SIGTERM , SIG_DFL);
        raise(SIGINT);
        }
//-------------------------------------------------------------------------------------------
void save(char nomefile[],char contenutofile[],int size);
/*Salva i dati che gli arrivano dal server in un file
        - nomefile - nome del file in cui salvare i dati in arrivo dal server
        - contenutofile - dati da salvare nel file
        - size - dimensione del file */

void save(char nomefile[],char contenutofile[],int size){
        int n;
        FILE *fp;

        fp = fopen(nomefile , "w");                     /* apriamo un file in scrittura */
        n = fwrite(contenutofile , 1 , size , fp);      /* e ci scriviamo i nostri dati */
        if (n < 0)
                printf("Errore nella scrittura del file\n");
        fclose(fp);
        }
//-------------------------------------------------------------------------------------------



//===========================================================================================
//                                                                      << M A I N >>
//===========================================================================================
int main(int argc, char *argv[]){
        struct sockaddr_in Serv;
        struct timeval inizio;
        int sockfd , OptVal , msglen , Fromlen , ris;
        int n , nread , nwrite , len , lung;              /* definizioni variabili */
        char file[MAXFILESIZE] , tipo;
        ErrorReplyHeader err;
        GetReplyHeader grep;
        GetRequestHeader greq;
        int *ris3;

        signal(SIGINT , segnale);
        _textattr(NORMAL);
        clrscr();
        if(argc!=5){                            /* controllo sull'immissione dati */
                usage(argv[0]);
                exit(1);
                }
        sockfd = socket(AF_INET, SOCK_STREAM, 0);     /* inizializziamo un nuovo socket */
        if (sockfd == SOCKET_ERROR) {                 /* e controlliamo se tutto è andato a buon fine */
                gotoxy(0,22);
                _textattr(RED);
                printf ("socket() failed, Err: %d \"%s\"\n" , errno , strerror(errno));
                _textattr(WHITE);
                exit(1);
                }

        OptVal = 1;     /* settiamo le opzioni del socket <riutilizzo dell'indirizzo> */
        ris = setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (char *)&OptVal, sizeof(OptVal));
        if (ris == SOCKET_ERROR)  {
                gotoxy(0,22);                /* controllo dell'errore */
                _textattr(RED);
                printf ("setsockopt() SO_REUSEADDR fallita, Errore: %d \"%s\"\n", errno,strerror(errno));
                _textattr(WHITE);
                exit(1);
                }

        memset( &Serv , 0 , sizeof(Serv) );
        Serv.sin_family = AF_INET;                              /* immissione dei parametri di connessione al server */
        if (inet_addr(argv[1]) == inet_addr("255.255.255.255"))
                return - 1;
        Serv.sin_addr.s_addr = inet_addr(argv[1]);
        Serv.sin_port = htons(atoi(argv[2]));

        ris = connect(sockfd, (struct sockaddr*) &Serv, sizeof(Serv));   /* tentiamo di connetterci al server */
        if (ris == SOCKET_ERROR)  {
                _textattr(RED);                                          /* controllo dell'errore */
                printf ("connect() fallita, Errore: %d \"%s\"\n",errno,strerror(errno));
                _textattr(WHITE);
                exit(1);
                }

        greq = makerequest(argv[3]);                    /* incapsulamento della richiesta GET */
        ris = writen(sockfd , &greq , sizeof(greq));    /* scrittura della richiesta sul server */
        if (ris == -1)                                  /* controllo dell'errore */
                return ris;

        ris = readn(sockfd , (char*) &grep , sizeof(grep.respons_code) , 0);  /* lettura della risposta alla GET dal server */
        if (ris == -1)
                return ris;
```

```
        tipo = grep.respons_code;       /* controlliamo il tipo del pacchetto appena ricevuto */
        if (tipo == 'G'){               /* ricevuto GET, quindi lettura della lunghezza del file */
                ris = readn(sockfd ,((char*) &grep) + sizeof(grep.respons_code) , (sizeof(GetReplyHeader) -
sizeof(grep.respons_code)) , 0);
                if (ris == -1){
                        _textattr(RED);
                        printf("Errore in ricezione\n");      /* controllo dell'errore */
                        _textattr(NORMAL);
                        return -1;
                        }
                }
        else
                if(tipo == 'E') {        /* ricevuto ERRORE, quindi lettura del messaggio d'errore */
                        ris = readn(sockfd , ((char*) &err) + sizeof(err.error_code) , (sizeof(ErrorReplyHeader) -
sizeof(err.error_code)) , 0);
                        _textattr(RED);
                        printf("Ricevuto errore: %s\n" ,err.msg);
                        _textattr(NORMAL);
                        return -1;
                        }
                else
                        return -1;

        lung = atoi(grep.bodysize);                     /* lunghezza del file */
        grafica_iniziale(argv[3] , argv[4] , lung);     /* disegniamo la tabella e i titoli */
        gettimeofday(&inizio , NULL);                   /*salviamo l'ora iniziale per il controllo del tempo nella parte grafica */
        ris = readn(sockfd , (char*) &file , lung - 1 , 1);  /* leggiamo il file di lunghezza <lung> */
       if (ris == -1)
                return ris;
        save(argv[4] , file , lung - 1);                /* salviamo i dati ricevuti in un file */
        close(sockfd);                                  /* chiudiamo il socket verso il server */

        _textattr(NORMAL);
        gotoxy(1 , 29);                 /* setto il font alla normalità, in modo da non influenzare */
        printf("\n");                   /* altre applicazioni */

        return 0;
        }
//============================================================================================== FINE MAIN =========




//=================================================== FUNZIONI X LA GRAFICA ================================================


void gotoxy(int x , int y){
        char es[50];
        char posx[25];
        char posy[25];

        sprintf(posx , "%d" , x);
        sprintf(posy , "%d" , y);
        es[0] = '\0';
        strcat(es , "\x1B[");
        strcat(es , posy);
        strcat(es , "d");
        strcat(es , "\x1B[");
        strcat(es , posx);
        strcat(es , "G");
        printf("%s" , es);
        }

void clrscr(void){
        printf("\x1B[2J"
                "\x1B[0d"
                "\x1B[0G");
        }


void _textattr(int attrib){
        if (attrib == YELLOW){
                _textattr(BROWN);
                _textattr(BOLD);
                }
        else{
                char es[50];
                char attr[25];
                sprintf (attr , "%d" , attrib);
                es[0] = '\0';
                strcat (es , "\x1B[");
                strcat (es , attr);
                strcat (es , "m");
                printf ("%s" , es);
                }
        }


int grafica_iniziale(char *richiesto, char *salvato, int filesize){
```

```
        int x,y;

        clrscr();
        _textattr(35);
        _textattr(BOLD);
        gotoxy(15,2);
        printf( "Progetto di Laboratorio"
                " di Programmazione di Reti\n");
        gotoxy(15,1);                                   /* titolo  del progetto*/
        printf("----------------------"
                "--------------------------\n");
        gotoxy(15,3);
        printf("----------------------"
                "--------------------------\n");

        _textattr(CYAN);
        gotoxy(30,4);
        printf("Prof. Fabio Panzieri\n");               /* All'attenzione di: */
        gotoxy(30,5);
        printf("Prof. Vittorio Ghini\n");

        _textattr(GREEN);
        _textattr(BOLD);                                /* file richiesto: <nome_file_dst> */
        gotoxy(13,7);                                   /*   salva come: <nome_file_rcv>   */
        printf("file richiesto: %s     -     "
                "salva come: %s",richiesto,salvato);

        gotoxy(25,8);
        _textattr(NORMAL);                              /* dimensione file : <n> byte */
        printf("dimensione file: %d byte", filesize);

        _textattr(BORDO_TABELLA);                       /* disegno i bordi della tabella */
        for (x = X_init , y = Y_init ; x < X_init + LARGHEZZA + 4 ; x++) {
                gotoxy(x , y);                              /*  ^^^ 4 = 2 caratteri di inizio + 2 di fine tabella */
                printf(" ");
                gotoxy(x , y + ALTEZZA_2 - ALTEZZA_1);
                printf(" ");
                }
        for (x = X_init , y = Y_init + 1 ; y < Y_init + ALTEZZA_2 - ALTEZZA_1 + 1 ; y++) {
                gotoxy(x , y);
                printf("  ");
                gotoxy(x + LARGHEZZA + 2 , y);
                printf("  ");
                }

        _textattr(COL_SCRITTE);             /* inserisco le scritte relative alla */
        _textattr(BG_BLACK);          /*  trasmissione dei dati             */
        gotoxy(15,14);
        printf("Percentuale:           0.00    %c",'%');   /* percentuale di file scaricato */
        gotoxy(15,15);
        printf("Velocità media:        0.00    B/s");      /* velocità in ricezione */
        gotoxy(15,16);
        printf("Tempo rimanente:       0       s");        /* secondi rimanenti per terminare il download */

        gotoxy(77,25);
        _textattr(BG_GREEN);         /* FORZA ITALIA! :) */
        printf(" ");
        _textattr(BG_WHITE);
        printf(" ");
        _textattr(BG_RED);
        printf(" ");
        gotoxy(0,25);
        _textattr(BG_GREEN);
        printf(" ");
        _textattr(BG_WHITE);
        printf(" ");
        _textattr(BG_RED);
        printf(" ");

        _textattr(YELLOW);
        _textattr(BG_BLACK);
        gotoxy(15,25);
        printf("A. Michetti");       /* Autori... */
        gotoxy(35,25);
        printf("R. Valenti");
        gotoxy(55,25);
        printf("A. Pegoraro");

        _textattr(NORMAL);          /* reimpostiamo il font alla normalità */
        printf("\n\n");
}


void disegna_asterischi (int filesize , int segmento){
        int riga;
        int colonna;

        for (riga = ALTEZZA_1 + 1 ; riga < ALTEZZA_2 ; riga++)
                for (colonna = (segmento * LARGHEZZA / filesize) - 1; colonna < ((segmento + 1) * LARGHEZZA / filesize) - 1 ;
colonna++) {
```

```
                        gotoxy (colonna + X_init + 3 , riga);      /* controllo per riempire tutta la tabella */
                        _textattr (COL_SCRITTE);
                        printf("%c" , 130);   /* il carattere 130 in ASCII corrisponde al quadratino */
                        }
        _textattr (WHITE);
        }

        void percentuale (int filesize , int byte){
        float perc;                     /*                        byte  *  100            */
        perc = byte * 100;              /*       percentuale = ------------------         */
        perc /= filesize;               /*                           filesize             */

        _textattr (COL_SCRITTE);
        gotoxy (32,14);                 /* Andiamo a scrivere la percentuale sullo schermo */
        printf ("%10.2f",perc);

        if (perc >= 100)
                {
                gotoxy (50,14);
                _textattr (BLINK);
                printf ("Download completato!!!\a");    /* tutto è andato a buon fine, almeno si spera! :) */
                _textattr (NORMAL);
                }
        _textattr (NORMAL);
        }

int velocita (struct timeval *inizio , struct timeval *fine , int segmento){
        float tempo , vel;

        vel = segmento;
        tempo = ((fine)->tv_sec) - ((inizio)->tv_sec) + 1;
        if (tempo < 0)                  /*                byte scritti   */
                tempo *= -1;            /* velocità = ----------------- */
        vel /= tempo;                   /*             tempo impiegato  */

        gotoxy(32 , 15);
        _textattr(COL_SCRITTE);              /* scrittura della velocità attuale su schermo */
        if (vel < 1000L)
                printf("%10.2f  B/s  " , vel);                      /* in  byte / sec */
        else
                printf("%10.2f kB/s  " , vel / 1000L);              /* in Kbyte / sec */
        _textattr(NORMAL);
        gotoxy(1,29);     /* posizioniamo il puntatore in fondo */
        return vel;
        }

void rimanenza (int segmento , int filesize , float velocita){
        float secondi , byte_scritti , finale;

        if (segmento != 0)
                byte_scritti = segmento;
        else
                byte_scritti = segmento + 1;         /*                      filesize * t_impiegato                 */
        secondi = byte_scritti / velocita;           /*  tempo_rimanente = ------------------------  - t_impiegato */
        if (secondi < 0)                             /*                              byte_scritti                   */
                secondi *= -1;
        finale = (secondi * filesize / byte_scritti) - secondi;

        gotoxy(32 , 16);
        _textattr(COL_SCRITTE);     /* scrittura del tempo mancante per terminare la trasmissione */
        printf("%10.0f" , - finale);      /* del file */
        _textattr(NORMAL);

        gotoxy(1,29);
        }
```

Codice sorgente del PROXY

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>  //per i segnali
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/errno.h>
#include "define.h"  //libreria user-defined
#include <sys/ipc.h> /*queste due librerie servono*/
#include <sys/shm.h> /*per le aree di memoria condivisa*/


//---------------impacchettatori--------------
SizeRequestHeader makesizerequest(char *nomefile);
//crea un pacchetto di tipo SizeRequestHeader con dentro il nomefile passato come parametro
GetRangeRequestHeader makegetrangerequest (char *nomefile, int begin, int end);
//crea un pacchetto di tipo GetRangeRequestHeader con dentro il nomefile, l'inizio e la fine passati come parametro
GetReplyHeader makegetreply(int lunghezza);
//crea un pacchetto di tipo GetReplyHeader con dentro la lunghezza passata come parametro
ErrorReplyHeader makeerrorreply(char *messaggio);
//crea un pacchetto di tipo ErrorReplyHeader con dentro il messaggio passato come parametro
//----------------------------------------
void alarm_h ();
//handler del segnale alarm:quando arriva un SIGALRM provvede a chiudere il processo padre(se non e' init) e successivamente se
stesso
void usage (char *nomefunzione);
/* Prende in input il nome della funzione (argv[0]) e scrive su stdout la modalita' di immissione parametri da riga di comando.
        Viene chiamata se il programma si accorge di un errore di immissione parametri.*/

int socketsetup(short int numero_porta_locale);
/* Inizializza il socket che servira' per ricevere le connessioni da parte dei client.
        Se una delle operazioni di inizializzazione fallisce la funzione restituisce la segnalazione su stdout del tipo di errore
        e -1 alla funzione chiamante, che a sua volta provvedera' a gestire la situazione(in questo caso terminera' il programma)
*/

int readn(int fd, char *ptr, int nbytes);
/* Funzione che assicura la lettura sicura da un filedescriptor di nbytes byte, andandoli a salvare in una stringa (ptr);
        con alcuni controlli la funzione cicla finché non riceve tutti i byte che ha chiesto, ritornando nbyte se tutto
        va a buon fine o -1 in caso di errore (insieme ad un messaggio di errore su stdout); restituisce invece 0 se
        nell'ultima read() contenuta al suo interno ha ricevuto 0 byte.*/

ssize_t writen (int fd, const void *buf, size_t n);
/* Funzione che assicura la scrittura sicura da un filedescriptor di n byte, prendendoli da un buffer puntato da *buf;
        con alcuni controlli la funzione cicla finché non riesce ad inviare tutti i byte specificati, ritornando n se tutto
        va a buon fine o -1 in caso di errore (insieme ad un messaggio di errore su stdout) */

int gestore_client (int client , char *nomefile );
/* controlla che il client gli mandi un byte che contenga il carattere 'G' (GET), dopodiché, se tutto e'  andato a buon fine,
        chiama gestore_get_request() che riceve im messaggio completo dal client.
        Il valore di ritorno della funzione e'  -1 se la richiesta dal client e'  diversa da quella che ci si aspetta ('G'),
        altrimenti il valore di ritorno della funzione gestore_get_request(). */

int gestore_server (int connectionfd,char *segmento);
/* Controlla il tipo di messaggio in arrivo dal server: se il messaggio ha come intestazione il carattere:
        - 'S': si tratta di una risposta alla SIZE e chiama gestore_size_reply();
        - 'R': si tratta di una risposta alla GETRANGE e chiama gestore_range_reply();
        - 'E': si tratta di un messaggio di errore e chiama gestore_error();
        - altrimenti: si e'  verificato un errore in trasmissione.
        Il valore di ritorno di questa funzione e'  il valore di ritorno di una delle funzioni chiamate sopra, tranne
        nell'ultimo caso, nel quale restituisce -1. */

int gestore_get_reply (int connectionfd,int lunghezza);
/* Crea un messaggio di risposta di tipo GET tramite la funzione make_get_reply() che restituisce un valore di tipo
GetReplyHeader,
        grande <lunghezza> byte e lo scrive su connectionfd, tramite una writen. Il valore di ritorno di questa funzione e'  -1
        in caso di errore della writen, 0 altrimenti. */

int gestore_size_reply (int connectionfd , SizeReplyHeader *srep);
/* Legge da connectionfd, tramite una readn(), la lunghezza del file specificato in precedenza tramite una richiesta di
        SIZE, salvandola in una variabile *srep di tipo SizeReplyHeader.
        Restituisce -1 in caso di errore in lettura, altrimenti il numero di byte del file. */

int gestore_range_reply (int connectionfd , GetRangeReplyHeader *grrep , char *segmento);
/* Legge, dapprima tramite una readn(), la lunghezza del segmento dati in arrivo dal server identificato da connectionfd,
        dopodiché esegue un'altra readn() di tanti byte quanti specificati nella precedente readn(), andandoli a salvare in
        una stringa <segmento> passata come parametro della funzione..
        La funzione restituisce 0 ni caso di esito positivo, -1 in caso di errore di una delle due read().*/

int gestore_error (int connectionfd , ErrorReplyHeader *err);
/* Stampa il messaggio di errore ricevuto dal server su stdout e torna -1.*/

int gestore_get_request (int client , GetRequestHeader *greq , char *nomefile);
/* Riceve un pacchetto di tipo GET dal client e salva su una stringa <nomefile> il nome del file che si vuole trasmettere,
        dopo averlo estrapolato da una variabile di tipo GetRequestHeader.
        La funzione ritorna 0 se tutto va a buon fine, -1 in caso di errore.*/
```

```
int gestore_getsize (char *filename , int server);
/* Dapprima crea un messaggio di tipo SizeRequestHeader tramite una makesizerequest(), poi lo invia al server tramite
        una writen; restituisce -1 in caso  di errore in trasmissione o 0 se tutto va bene. */

int mandaerrore(char *msg ,int clientfd);
/* Crea un pacchetto di tipo ErrorReplyHeader, dopodiché lo invia al file descriptor <clientfd> tramite una writen.
        Restituisce 0.*/

int forka (char *filename , int filesize , struct sockaddr_in *server , int clientfd , int nserver);
/* Questa funzione e'  il nocciolo del nostro programma. Il vero lavoro della forka() e'  quello di generare un certo numero di
        processi-segmento tramite una fork(), ognuno dei quali lavorera' per conto proprio chiedendo al server un segmento di file
che
        poi andremo ad inviare al client. L'ordinamento di questi avviene grazie a dei controlli implementati tramite
        shared memory: proprio nelle prime righe di questa funzione trovimo infatti l'inizializzazione di due variabili condivise.
        La prima e'  un puntatore ad una area di memoria (*shpointer) che servira' per comunicare il numero del segmento da
inviare.
        l'altra e' un array di char (status) che servira' per memorizzare lo stato dei server.
        Tutto l'array di stato (status) verra' settato a '0', in modo da dichiarare, almeno per il momento, che tutti i server
sono liberi.
        Il cilco for che verra' iterato tante volte quanti sono i segmenti in cui viene diviso il file trasmesso
        dai server; appena si entra in questo ciclo viene fatta una fork():
                - il processo padre fungera' da "limitatore":testera' se la connesione al client e' attiva tramite una select,
dopodiché testera'
                  se il server e'  occupato; in caso contrario prima di concedere la connessione controllera' che non ci siano
troppi processi attivi:
                  a questo punto puo' lasciare che la funzione possa generare altri processi-segmento.
                - il processo-segmento e'  costituito da un ciclo loopback che controlla se esiste qualche server attivo; una
volta trovato
                        cerca di instaurare una connessione e una volta fatta chiamera' makegetrangerequest(), la quale crea un
pacchetto
                        di tipo GetRangeRequestHeader che verra' quindi scritta sul server tramite una writen() per richiedere un
certo
                        pezzo di file. A questo punto aspetta il serve mandi cio' che gli e' stato appena chiesto.
                        Ricevuto il segmento di file il processo deve aspettare che sia il suo turno per scriverlo sul
                        client. Per testare quando e'  arrivato il turno aspetta che *shpointer sia uguale ad i(il numero del
segmento gestito).
                        E' stato introdotto anche un controllo in modo che si possano auto-terminare (timeout) tutti quei processi
che rimangono
                        zombie una volta che il client ha terminato l'esecuzione in maniera irregolare(crash).
*/

//=============================================================================================================================================
//                                                      MAIN
//=============================================================================================================================================

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

struct sockaddr_in Local , Server , servers[MAXSERVERS] , Cli;
int nserver , listenfd , connfd , i , k , filesize , sockfd , ris;
unsigned int Len;
char filename[FILENAMELEN];
SizeRequestHeader sreq;
pid_t pid;

if ((argc < 8) || (argc > 2+(2*MAXSERVERS)) || (argc % 2)) {                    // controlli e assegnazione degli argomenti
passati al programma
        usage(argv[0]);
        return -1;
        }
nserver = (argc - 2) / 2;                                                       //calcolo del numero dei server
for (i = 0 ; i < nserver ; i++){                                                /*Ciclo di inizializzazione array "servers" :*/
          memset(&Server , 0 , sizeof(Server));                                 /*i server passati da riga di comando vengono
salvati in modo*/
          Server.sin_family = AF_INET;                                          /*da poter essere ripresi per reinstaurare una
connessione*/
          if (inet_addr(argv[(i * 2) + 2]) == inet_addr("255.255.255.255"))
                return -1;
          Server.sin_addr.s_addr = inet_addr(argv[(i * 2) + 2]);
          Server.sin_port = htons(atoi(argv[(i * 2) + 3]));
          servers[i] = Server;
         }


                                                                                //setup degli handler dei segnali
signal(SIGHUP ,  SIG_DFL);
signal(SIGINT ,  SIG_DFL);
signal(SIGTERM , SIG_DFL);                                                      /*SIGCHLD viene ignorato in modo da poter
permettere ai processi*/
signal(SIGCHLD , SIG_IGN);                                                      /* figli di "morire in pace" e quindi non
diventare "zombie"*/

listenfd = socketsetup(atoi(argv[1]));                                          /*Configurazione porta listen locale, se si
verifica*/
if (listenfd == -1)                                                             /*un errore, questo verra' segnalato dalla
funzione stessa*/
        return -1;                                                              //Qui, se la socketsetup() fallisce, il programma
esce.
for( ; ; ){                                                                     //Inizia il ciclo cinfinito che accetta le
connessioni dai client
        printf ("accept()\n");
```

```
        fflush(stdout);
        do {
                Len = sizeof(Cli);
                memset(&Cli , 0 , sizeof(struct sockaddr_in) );
                connfd = accept(listenfd , (struct sockaddr*) &Cli , &Len);     //Aspetta una connessione da parte di un client
                } while ((connfd < 0) && (errno == EINTR));
        if (connfd < 0){
                printf ("accept() failed, Err: %d \"%s\"%d\n" , errno , strerror(errno));
                fflush(stdout);
                return -1;
                }                                                               //se l'accept() fallisce il programma termina.

        printf("fork()\n");
        fflush(stdout);
        pid = fork();                                                           //Duplicazione dei processi in modo da poter
accettare altre connessioni
        if(pid < 0) {
                close(listenfd);
                printf ("fork() failed, Err: %d \"%s\"\n" , errno , strerror(errno));
                fflush(stdout);
                return -1;                                                      //se duplicazione fallisce chiusura del programma.
                }

        if ( pid == 0 ) {                                                       //inizio del codice del processo figlio
                close(listenfd);
                printf("\t\tCreato processo %d\n",getpid());
                fflush(stdout);
                ris = gestore_client(connfd , filename);                        //Leggo il nome del file dal client
                if (ris < 0) {
                        mandaerrore("Errore di comunicazione nome file" , connfd);//se c'e' un errore avviso il client
                        exit(1);                                                // e killo il processo;
                        }
                k=0;                                                            // k e' il contatore degli errori, se>MAXERR il
programma termina.
                for(i = 0 ; ; i++){                                             //Inizio un ciclo infinito(serve per implementare
loopback)
                        sockfd = socket(AF_INET , SOCK_STREAM , 0);
                        if (sockfd < 0) {
                                fprintf (stderr, "socket() failed, Err: %d \"%s\"\n" , errno ,strerror(errno));
                                fflush(stderr);
                                continue;}                                      //se fallisco faccio loopback (riprovo da capo)

                        ris = connect(sockfd , (struct sockaddr *) &servers[i % nserver] , sizeof(servers[i % nserver]));
                        if (ris < 0){
                                        printf("Errore di connessione con il %d° server\n" , i % nserver + 1);
                                        fflush(stdout);
                                        continue;                               //se fallisco faccio loopback
                                      }
                        ris = gestore_getsize(filename , sockfd);               // provo ad inviare la richiesta GETSIZE al server
scelto
                        if (ris < 0){                                           //se fallisco
                                k++;                                            //incremento il contatore degli errori
                                if (k > MAXERR) {                               //se sono arrivato al numero massimo di errori...
                                        mandaerrore("Errore di comunicazione con i servers\n", connfd);
                                        exit(1);                                //... lo comunico e killo il processo.
                                        }
                                continue;                                       //loopback nel caso k<MAXERR
                                }
                        filesize = gestore_server(sockfd , NULL);               //provo a leggere la risposta dal server
                        if (filesize == -2) {                                   //se risponde con un errore...
                                        mandaerrore("Richiesta non valida\n" , connfd);
                                        exit(1);                                //termino il programma
                                        }
                        if (filesize == -1){                                    //se fallisco
                                k++;                                            //incremento contatore degli errori
                                if (k > MAXERR) {                               //se sono arrivato al numero massimo di errori...
                                        mandaerrore("Connessione temporaneamente skifosa\n" , connfd);
                                        exit(1);                                //... lo comunico e termino il programma.
                                        }
                                continue;                                       //loopback nel caso k<MAXERR
                                }
                        if ((filesize >= 1000000000L) || (filesize >= MAXFILESIZE)){ //qui ho gia' ricevuto il filesize.Controllo
se e' nella "norma"
                                mandaerrore("File richiesto troppo grande\n" , connfd); //se e' troppo grande segnalo l'errore
                                exit(1);                                        //...e killo il processo
                                }
                        printf ("Nome file: %s   -   Lunghezza file: %d byte\n" , filename , filesize);
                        fflush(stdout);
                        break;                                                  // esco dal ciclo di loopback
                        }
                ris=gestore_get_reply(connfd , filesize);                       //mando la dimensione del file al client in modo
da metterlo in attesa
                if (ris==-1) exit(1);                                           //se fallisco killo il processo
                printf("%11s\t %11s\t %11s\t %11s\t\n","PROCESS","CREATO","RICEVUTO","INVIATO");
                forka (filename , filesize - 1 , servers , connfd , nserver);   //chiamo il generatore di processi per la gestione
dei singoli segmenti
                printf("   Il processo-generatore %d ha terminato,figli orfani adottati da init(pid=1)\n",getpid());
                fflush(stdout);
                close(connfd);                                                  //chiudo la connessione con il client
                raise(SIGINT);                                                  //mando a me stesso il segnale di terminazione
```

```
                }
        if (pid > 0)                                                    //inizio codice del processo padre
                close(connfd);                                          //chiudo la connessione che sta usando il figlio

        }                                                               //fine ciclo infinito
}                                                                       //fine main (questa riga non verra' mai eseguita
causa ciclo infinito)

//=============================================================================================================================
=
//=============================================================================================================================
=
```

```
//=========================================FUNZIONI=============================================================================

int forka (char *filename , int filesize , struct sockaddr_in *server , int clientfd , int nserver){
        int i ,ris ,sockfd , k , shmid , *shpointer , shstatusid;
        pid_t pid;
        char segment[MAXSEGMENTSIZE - 1] , *status,testedstatus[nserver];
        GetRangeRequestHeader grrq;
        struct shmid_ds buf;
        fd_set rset;

        buf.shm_perm.uid = getuid();                                    /*setto su una struttura semid_ds i permessi*/
        buf.shm_perm.gid = getgid();                                    /*in lettura e scrittura     */
        buf.shm_perm.mode = (unsigned short)511;                        /*su una variabile condivisa */
        shmid = shmget(0 , sizeof(int) , IPC_CREAT);                    //Richiedo un area di memoria condivisa
        ris = shmctl(shmid , IPC_SET , &buf);                           //setto sull'area di memoria condivisa i permessi
che avevo messo in buf
        shpointer = shmat(shmid , 0 , 0);                               // Attach della memoria, ottengo un puntatore
        *shpointer = 0;                                                 /*inizializzo il puntatore a 0,servira' per sapere
quale segmento*/
                                                                        /*ha gia' scritto sul client*/
       shstatusid = shmget(0 , nserver - 1 , IPC_CREAT);                //Richiedo un array di memoria condivisa grande
nserver-1
        ris = shmctl(shstatusid , IPC_SET , &buf);                      //setto i permessi
        status = shmat(shstatusid , 0 , 0);                             /*Ottengo il puntatore, servira' per memorizzare
                                                                        /*lo stato dei server (1=occupato 0=libero)*/
        signal(SIGALRM,alarm_h);
        for (i = 0 ; i < nserver ; i++)
                 *(status+i) = '0';                                     //Inizializzo tutti i valori a 0 (server liberi)
        for (i = 0 ; i <= (filesize/MAXSEGMENTSIZE) ; i++){             //Ciclo per creare tutti i segmenti
          pid = fork();
                if (pid == 0) {                                         //processo figlio(inizio vita di un processo-
segmento)
                        alarm(60);                                      //timeout per richieste al server
                        for(; ;k++){                                      //inizio ciclo infinito per loopback
                                sockfd = socket(AF_INET , SOCK_STREAM , 0);     //creo un socket
                                if (sockfd < 0)
                                        continue;                       //se fallisco faccio loopback

                                 for(;;){                               //ciclo infinito per test dei server;
                                         if (testedstatus[k%nserver]=='x') {    //se il server e' error-prone
                                                k++;                    //al prossimo ciclo cambio server
                                                if (k>60000)            //se k e' troppo grossa (quindi ho girato tutti i
server piu volte)...
                                                        for (k=0;k<nserver;k++)
                                                                testedstatus[k]=' ';//resetto lo status di error-prone
                                                continue;               //continuo il test dei server
                                                }
                                        if (*(status + (k % nserver)) == '0'){ //se oltre ad essere "affidabile" e' libero
                                          *(status + (k % nserver)) = '1';      //lo occupo e ... provo una connessione
                                          ris = connect(sockfd , (struct sockaddr *) &server[k % nserver] , sizeof(server[k %
nserver]));
                                                if (ris == -1){         //se fallisce
                                                        *(status + (k % nserver)) = '0';//libero il server
                                                        testedstatus[k%nserver]='x';//lo setto come error-prone(probabilmente non
esiste)
                                                        k++;            //al prossimo ciclo cambio server
                                                        continue;       //loopback
                                                        }
                                                else
                                                        break;          //esco dal test dei server
                                                }
                                        }
                                if (i < filesize / MAXSEGMENTSIZE)      //se non e' l'ultimo segmento creo una richiesta
range "normale"
                                        grrq = makegetrangerequest(filename , i * MAXSEGMENTSIZE , (i + 1) * MAXSEGMENTSIZE - 1);
                                else                                    //altrimenti la creo "ridotta" agli ultimi byte
rimanenti
                                        grrq = makegetrangerequest(filename , i * MAXSEGMENTSIZE , filesize - 1);
                                ris = writen(sockfd , (char*) &grrq , sizeof(grrq));//porvo a mandare la richiesta al server
```

```
                                if (ris != sizeof(grrq)){                       //se c'e' qualche errore..
                                       *(status + (k % nserver)) = '0';         //libero il server
                                        testedstatus[k%nserver]='x';            //lo setto come error-prone
                                        continue;                               //loopback
                                        }
                                ris = gestore_server(sockfd , segment);         //provo a leggere la risposta dal server
                                if (ris == -1){                                 //se c'e' qualche errore...
                                       *(status + (k % nserver)) = '0';         //libero il server
                                        testedstatus[k%nserver]='x';            //lo setto come error-prone
                                        continue;                               //loopback
                                        }
                                alarm (0);                                      //reset del timeout
                                printf("|\t%d\t|\t\t|\t%d(%d)\t|\t\t|\n" ,getppid(),i,k%nserver);//Qui il processo-segmento ha in
mano il suo segmento
                                fflush(stdout);
                                *(status + (k % nserver)) = '0';                //libero il server
                                break;                                          //quindi esco dal ciclo di loopback
                                }
                        alarm(30);                                              //setto un timeout
                        while (*shpointer != i);                                /*testa la variabile condivisa per sapere se e'
possibile*/
                                                                                /*inviare al client,SIGALRM killa processo se
scade timeout*/
                        alarm(0);                                               //disttivo timeout
                        if (i < filesize / MAXSEGMENTSIZE){                     //se non e' l'ultimo segmento ...
                                    ris = writen(clientfd , (char*) &segment , MAXSEGMENTSIZE);//scrivo un segmento di dimensioni
"normali"
                               if (ris != MAXSEGMENTSIZE)                       //se si verificano errori...
                                            exit(1);                            //killo il processo
                                }
                        else {                                                  //se e' l'ultimo segmento..
                                ris = writen(clientfd , (char*) &segment , filesize % MAXSEGMENTSIZE);//scrivo un segmento della
dimensione rimanente
                                if (ris != filesize % MAXSEGMENTSIZE)           //se c'e' qualche errore
                                           exit(1);                             //killo il processo
                                }
                        printf("|\t%d\t|\t\t|\t\t|\t%d\t|\n" ,getppid(), i);//qui il segmento e' stato inviato... il processo puo'
crepare
                        fflush(stdout);
                        *shpointer += 1;                                        //ultimo desiderio:passare il compito al processo
successivo :)
                        exit (1);                                               //killo il processo
                        }
                else {                                                          //codice processo padre
                        printf("|\t%d\t|\t%d\t|\t\t|\t\t|\n" ,getpid(),i);      //comunico la creazione di un nuovo processo
                        fflush(stdout);
                /*      FD_ZERO(&rset);                                          //resetto la struttura
                        FD_SET(clientfd,&rset);                                 //setto la connessione nella struttura
                        ris=select(clientfd+1,NULL,&rset,NULL,0);               //testo se la connessione con il client e' ancora
viva
     (?)                printf("ris=%d\n",ris);
                        if (ris<=0){
                                printf("\t\tProcesso %d termina anomalmente\n",getpid());
                                fflush(stdout);
                                exit(1);                                        //se e' crashata killo il processo
                                } */
                        for (k = 0 ; ; k++)                                     //ciclo infinito per testare la disponibilita' dei
server
                              if( *(status + (k % nserver)) == '0')             //se uno e' disponibile...
                                  if (i<(*shpointer)+(nserver*2)-1)             //e non ci sono troppi processi-segmento in attesa
                                  break;                                        //allora esco dal ciclo e torno a creare un nuovo
processo-segmento.

                      }
                }
        }

//-------------------------------------------------------------------------------------------------------------------------

void usage (char *nomefunzione){
       printf("Error: invalid number of args;\n");
       printf("Usage: %s listenport server1 port1 server2 port2 server3 port3 [server_n port_n]\n" , nomefunzione);
       }

int socketsetup(short int numero_porta_locale)
{
       int listenfd;
       int OptVal;
       int ris;
       struct sockaddr_in Local;

       printf ("socket()\n");
       fflush(stdout);
       listenfd = socket(AF_INET , SOCK_STREAM , 0);                            //inizializzo un socket per la connessione col
peer
       if (listenfd < 0) {                                                      // controllo se la socket() ha fallito
               fprintf (stderr, "socket() failed, Err: %d \"%s\"\n" , errno ,strerror(errno));
               fflush(stderr);
               return -1;                                                       //se non e'  andata a buon fine termino
l'esecuzione tornando -1
```

```
                }
        OptVal = 1;
        printf ("setsockopt()\n");
        fflush(stdout);                                                 //aggiungo al socket appena inizializzato delle
opzioni
        ris = setsockopt(listenfd , SOL_SOCKET , SO_REUSEADDR , (char *) &OptVal , sizeof(OptVal));
        if (ris < 0){                                                   //controllo se ci sono stati dei problemi nella
setsockopt()
                fprintf (stderr , "setsockopt() SO_REUSEADDR failed, Err: %d \"%s\"\n" , errno ,strerror(errno));
                fflush(stderr);
                return -1;
                }                                                        //esco e ritorno -1 se ci sono stati dei
problemi
        printf("porta locale %d\n" , numero_porta_locale);
        fflush(stdout);
        memset ( &Local , 0 , sizeof(Local));                            //inizializzo a 0 la struttura Local
        Local.sin_family = AF_INET;
        Local.sin_addr.s_addr = htonl(INADDR_ANY);                       //Salvo nella Local i dati della connessione
verso il client
        Local.sin_port = htons(numero_porta_locale);
        printf ("bind()\n");
        fflush(stdout);
        ris = bind(listenfd , (struct sockaddr*) &Local , sizeof(Local));        //colleghiamo al socket l'indirizzo definito da
Local
        if (ris < 0){                                                    // controlliamo se tutto e'  andato a buon fine
                fprintf (stderr , "bind() failed, Err: %d \"%s\"\n" , errno , strerror(errno));
                fflush(stderr);
                return -1;                                               //se la bind() restituisce un errore esco e
torno -1
                }
        printf("listen()\n");
        fflush(stdout);                                                  //mi metto in ascolto sul socket aspettando una
connessione
        ris = listen(listenfd , 1);
        if (ris<0){                                                      //controllo se tutto e'  andato a buon fine
                fprintf ( stderr, "listen() failed, Err: %d \"%s\"\n",errno,strerror(errno));
                fflush(stderr);
                return(-1);
                }                                                        //esco e torno -1 se qualcosa e'  andata male
        return(listenfd);
        }


int readn(int fd, char *ptr, int nbytes){
        int nleft,nread;

        nleft = nbytes;                                                   //all'inizio i byte mancanti sono il numero di
byte richiesti
        while(nleft > 0){                                                 //ripeto finché i byte mancano non siano 0
                nread = read(fd , ptr , nleft);                           //leggo nleft byte
                if(nread < 0){                                            //se l'esito della read() e'  negativo..
                        char msg[2000];
                        sprintf(msg , "readn: errore in lettura [result %d] :" , nread);
                        perror(msg);
                        return(-1);                                       //...stampo un errore e torno -1
                        }
                else {
                        if(nread == 0) {                                  //...se ne ho letti solo 0 allora ritorno 0;
                                return(0);
                                break;
                                }
                        }                                                 //altrimenti il n di byte mancanti sono
decrementati del n di quelli appena letti
                nleft -= nread;                                           //e sposto il puntatore di n byte letti
                ptr   += nread;
                }
        return(nbytes);                                                   //torno il numero di byte letti
        }

ssize_t writen (int fd, const void *buf, size_t n){
        size_t nleft;
        ssize_t nwritten;
        char *ptr;

        ptr = (char*)buf;                                                 //inizializzo un puntatore al buffer da cui
leggiamo
        nleft = n;                                                        //i byte mancanti sono quelli passati come
argomento (n)
        while (nleft > 0) {                                               //ripeto finché non ho scritto tutti i byte
                if ( (nwritten = write(fd, ptr, nleft)) <= 0) {           //scrivo nwritten byte e controllo se e'
andata male
                        if (errno == EINTR)                               //controllo se e'  avvenuto un errore dovuto ad
un interrupt
                                nwritten = 0;                             //se e'  così ricomincia tutto daccapo
                        else {
                                fprintf (stderr, "write() failed, Err: %d \"%s\"\n",errno,strerror(errno));
                                fflush(stderr);
                                return(-1);                               //se qualcosa di diverso e'  andato male torno
-1
                                }
```

```
                        }
                        nleft -= nwritten;                                          //diminuisco i byte da scrivere del numero di
byte scritti
                        ptr += nwritten;                                            //incremento il puntatore al buffer
                }
                return(n);                                                          //ritorno il numero di byte scritti
        }



//=============================================================== IMPACCHETTATORI ======================
//con queste funzioni costruisco pacchetti riconosciuti da server e client
//-------------------------------------------------------------------------------------------------------------------------
SizeRequestHeader makesizerequest(char *nomefile){                                  /* costruisco un pacchetto per il server per
richiedere.. */
        SizeRequestHeader header;                                                   /* ..la dimensione in byte del file *nomefile
*/

        memset(&header , 0 , sizeof(header));
        header.request_code = 'S';                                                  //inserisco come header il carattere 'S' di
SIZE
        sprintf(header.filename , "%s" , nomefile);                                 //inserisco il filename nel pacchetto
        return header;                                                              //restituisco come val di ritorno il pacchetto
        }
      /*
                 ___  ___ ___ ___ ___ ___ ___ ___ ___ ___ ___     ___
                |   ||   |   |   |   |   |   |   |   |   |...|   |          Esempio di pacchetto di tipo SIZE
                |'s'||'n'|'o'|'m'|'e'|\0 |\0 |\0 |\0 |\0 |\0 |...|   |
                |___||___|___|___|___|___|___|___|___|___|___|...|___|
                ^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      1byte di tipo                                 255 byte per il nome del file   */


GetRangeRequestHeader makegetrangerequest (char *nomefile, int begin, int end){     /* costruisco un pacchetto per il server per
richiedere.. */
        GetRangeRequestHeader header;                                               /* ..un pezzettino di *nomefile, a partire da
begin fino..*/
                                                                                    /* a end */
        memset(&header , 0 , sizeof(header));
        header.request_code = 'R';                                                  //il primo carattere e'  'R'
        sprintf(header.filename , "%s" , nomefile);                                 //poi aggiungo massimo 255 caratteri
contenenti il nome del file
        sprintf(header.start , "%d" , begin);                                       //l'inizio del pezzettino
        sprintf(header.end , "%d" , end);                                           //la fine del pezzettino
        return header;                                                              // quindi torno il pacchetto di tipo
GetRangeRequestHeader
        }

GetReplyHeader makegetreply(int lunghezza){                                         /* costruisco un pacchetto per il client per
inviare.. */
        GetReplyHeader header;                                                      /* ..il file di lunghezza <lunghezza> byte*/

        memset(&header , 0 , sizeof(header));
        header.respons_code = 'G';                                                  //Primo byte = 'G' (GET)
        sprintf(header.bodysize , "%d" , lunghezza);                                /*inserisco poi la lunghezza del file; stara'
poi a lui..*/
        return header;                                                              /*..scaricare tanti byte quanti specificati*/
        }


ErrorReplyHeader makeerrorreply(char *messaggio){                                   /*tipo particolare di pacchetto: contiene il
tipo di errore*/
        ErrorReplyHeader header;                                                    /*verra' inviato al client in caso di errore
*/

        memset(&header , 0 , sizeof(header));
        header.error_code = 'E';                                                    //tipo di mesaggio: 'E' (ERROR)
        sprintf(header.msg , "%s" , messaggio);                                     //inserisco poi il tipo di errore ricevuto dal
server
        return header;
        }
//=============================================================================================================


//============================================================================== GESTORI I/O ==================

int gestore_client (int client , char *nomefile ){                                  //gestisco le richieste da parte dei client
        int ris;
        GetRequestHeader greq;

        ris = readn(client , (char*) &greq , 1);                                    //leggo il primo byte per capire l'entita' del
pacchetto
       if (ris == -1)                                                               //controllo un eventuale errore in lettura
                return -1;
        if (greq.request_code == 'G')                                               //se e'  di tipo GET chiamo
gestore_get_request()
                ris = gestore_get_request (client , (GetRequestHeader *) &greq , nomefile);
        else
                ris= -1;                                                            //altrimenti si e'  verificato un errore e
ritorno -1
```

```
        return ris;                                                     //ritorno il valore di ris
        }


int gestore_server (int connectionfd,char *segmento){                   //gestisco la ricezione dei pscchetti in
arrivo dal server
        int ris;
        typedef union {                                                 /*Mi definisco un tipo di unione che puo'
assumere le 3*/
                GetRangeReplyHeader grrep;                              /*tipologie di pacchetto (GET , SIZE ,
ERROR)*/
                SizeReplyHeader srep;
                ErrorReplyHeader err;
                } Risposte;
        Risposte risposta;

        ris = readn(connectionfd , (char *) &risposta , 1);              //leggo il primo byte per capire l'entita'
del pacchetto
        switch (risposta.grrep.respons_code){
                case 'S': ris = gestore_size_reply(connectionfd , (SizeReplyHeader *) &risposta);
                        break;                                          //...se e'  'S' chiama gestore_size_reply()
                case 'R': ris = gestore_range_reply(connectionfd , (GetRangeReplyHeader *) &risposta , segmento);
                        break;                                          //...se e'  'R' chiama gestore_range_reply()
                case 'E': ris = gestore_error(connectionfd , (ErrorReplyHeader *) &risposta);
                        break;                                          //...se e'  'E' chiama gestor_error()
                default: ris = -1;                                      //altimenti c'e'  stato un errore di
impacchettamento o
                }                                                       //...di trasmissione dal server
        return ris;
        }

//=============================================================================================================

int gestore_get_reply (int connectionfd,int lunghezza){
        int ris;
        GetReplyHeader risposta;

        risposta = makegetreply(lunghezza);                             //creo il pacchetto
        ris = writen(connectionfd , (char *) &risposta , sizeof(risposta));          //lo invio
        if (ris != sizeof(risposta))                                    //se sbaglio
                return -1;                                              //ritorno -1 alla funzione chiamante
        return 0;                                                       //altrimenti 0
        }



int gestore_size_reply(int connectionfd , SizeReplyHeader *srep){
        int ris , size , nbyte;
        SizeReplyHeader srh;

        ris = readn(connectionfd , ((char *) srep + sizeof(srep->respons_code)) , sizeof(SizeReplyHeader) - sizeof(srep-
>respons_code));
        if (ris != sizeof(srep->filesize))                              //leggo la rimanenza del pacchetto
                return -1;                                              //se sbaglio ritorno -1
        nbyte = atoi(srep->filesize);
        return nbyte;                                                   //altrimenti il valore in int contenuto nel
corpo dell'header
        }



int gestore_range_reply(int connectionfd , GetRangeReplyHeader *grrep , char *segmento){
        int ris , lunghezzaint;
        ris = readn(connectionfd , ((char *) grrep + sizeof(grrep->respons_code)) , sizeof(GetRangeReplyHeader) - sizeof(grrep-
>respons_code));
        if (ris != sizeof(GetRangeReplyHeader) - sizeof(grrep->respons_code))        //leggo il rimanente del pacchetto
                return -1;                                              //se sbaglio torno -1
        ris = readn(connectionfd , segmento , atoi(grrep->bodysize));               //mi metto in attesa di n byte (n contenuto
nel corpo del precedente header
        if (ris != atoi(grrep->bodysize))                               //se sbaglio..
                return -1;                                              //torno -1
        return 0;
        }


int gestore_error(int connectionfd , ErrorReplyHeader *err){
        printf("Ricevuto un errore dal server\n");
        fflush(stdout);
        return -2;                                                      //segnalo l'errore in stdout e alla funzione
chiamante
        }

int gestore_get_request(int client , GetRequestHeader *greq , char *nomefile){
        int ris;

        ris = readn(client , ((char *)greq + sizeof(greq->request_code)) , sizeof(GetRequestHeader) - sizeof(greq->request_code));
        if (ris != sizeof(GetRequestHeader) - sizeof(greq->request_code))            //leggo il pacchetto dal client
                return -1;                                              //se sbaglio torno -1
        sprintf(nomefile , "%s" , greq->filename);                      //alrimenti metto il risultato sul puntatore
passato come parametro
```

```
        return  0;
        }

int gestore_getsize(char *filename , int server){
        SizeRequestHeader sreq;
        int ris;

        sreq = makesizerequest(filename);                                       //creo il pacchetto
        ris = writen(server , &sreq , sizeof(sreq));                            //lo invio
        if(ris != sizeof(sreq))                                                 //se sbaglio...
                return -1;                                                      //torno -1
        return 0;
        }

int mandaerrore(char *msg ,int clientfd){
        int ris;
        ErrorReplyHeader erh;
        erh = makeerrorreply(msg);                                              //creo il pacchetto
        ris = write(clientfd , (char*) &erh , sizeof(erh));                     //lo invio
        return ris;                                                             /*utile solo a sapere se il pacchetto e'
stato inviato*/
        }                                                                       /*la connessione verra' chiusa dalla funzione
chiamante*/


 void alarm_h (){
        if (getppid()!=1){                                                       //il chiamante non e' stato "adottato" da
init...
            printf("\tUn segmento fa terminare il processo %d per timeout\n",getppid());
            fflush(stdout);
            kill(getppid(),SIGINT);                                             //killo il padre in modo da non fargli
generare alri figli
            }
         exit(1);                                                           //killo il chiamante
        }
//-------------------------------------------------------------------------------------------------



//=================================================================================================================================
//                                              FINE FILE
//=================================================================================================================================
```