

Esercitazioni Socket

Ping-Pong TCP

Si realizzi una coppia di processi:

- il primo e' un server che pubblica un socket (SOCK_STREAM, TCP) ad un certo indirizzo (porta passata come parametro sulla riga di comando) e attende connessioni su tale socket. Ogni volta che accetta una connessione, legge un messaggio di un byte (e.g. un carattere) e lo riscrive sul socket cosi' com'e' sfruttando la bidirezionalita' della comunicazione TCP. Appena riscritto il carattere il server chiude la connessione e si predispone ad accettare un'altra connessione
- il secondo e' un client che effettua un ciclo di n iterazioni. Ad ogni iterazione si connette al server, gli invidia un carattere e legge un carattere come risposta. Immediatamente prima e immediatamente dopo il ciclo il cliente prende il tempo ed alla fine stampa il tempo medio per ognuna delle connessioni come tempo totale (i.e. tempo dopo il ciclo - tempo prima del ciclo) diviso per le iterazioni.
Per leggere il tempo (come secondi trascorsi dall' *epoca*) si puo' utilizzare la chiamata di sistema `time_t time(time_t *)`.
- `time(NULL)` restituisce i secondi trascorsi dall'epoca.
Alternativamente si puo' utilizzare la chiamata `int gettimeofday(struct timeval *tv, struct timezone *tz)` che restituisce valori piu' fini del tempo (fino al microsecondo, vedi `man gettimeofday`).

Si consideri la variante in cui la connessione viene aperta una volta sola e nel ciclo si effettuano semplicemente (ad ogni iterazione) la scrittura e la lettura del messaggio di un carattere

Calcolatrice TCP

Si realizzi una coppia di processi:

- il primo e' un server che pubblica un socket (SOCK_STREAM, TCP) ad un certo indirizzo (porta passata come parametro sulla riga di comando) e attende connessioni su tale socket. Ogni volta che accetta una connessione, legge un messaggio nel seguente formato:

operando1|operando2|operazione

dove `operando1` e `operando2` sono due numeri, `operazione` è il simbolo di una delle quattro operazioni (+, -, *, /) e `|` è un separatore; il server esegue l'operazione specificata e scrive il risultato sul socket cosi' com'e' sfruttando la bidirezionalita' della comunicazione TCP. Appena scritto il risultato il server chiude la connessione e si predispone ad accettare un'altra connessione

- il secondo e' un client che effettua un ciclo di n iterazioni. Ad ogni iterazione chiede in input due numeri ed un'operazione, si connette al server, gli invidia un messaggio (vedi server) e legge il risultato come risposta e lo stampa a video.

Server multithread

Si realizzi un server che incapsula una variabile di tipo intero ed accetta richieste di operazioni di incremento/decremento della variabile di valori arbitrari (cioè accetta richieste contenenti un valore N che va sommato alla variabile incapsulata e che può essere negativo).

Le richieste devono essere inviate utilizzando socket `SOCK_STREAM`, TCP.

Il server deve essere realizzato in modo multithreaded, ovvero deve essere utilizzato un thread separato per ognuna delle connessioni. (si consideri la possibilità di attivare un nuovo thread per il trattamento di ogni singola connessione, oppure di attivare un thread di un insieme di thread pre-forkati e in attesa di essere attivati per una nuova connessione. In quest'ultimo caso, il thread che termina il trattamento di una singola connessione non termina, ma si rimette in attesa di essere riattivato.)

Il server deve mantenere una lista delle operazioni effettuate sulla variabile incapsulata (giornale delle modifiche). A tale scopo va previsto che i messaggi di richiesta di servizio inviati al server contengano anche il nome della macchina da cui sono partiti e il pid del processo cliente che li ha inviati (informazioni inserite nel messaggio a carico del cliente, quindi).

Vogliamo che sia previsto un ulteriore thread che, ogni N secondi (N costante `#define N 10`) stampa a video il contenuto del giornale.

Server multithread UDP

Si realizzi una versione del server dell'esercizio precedente utilizzando UDP come protocollo per i socket, ovvero utilizzando il modello di comunicazione connection-less

Copia file con socket

Si realizzino una coppia di programmi cliente/servente che, utilizzando i socket come mezzo di trasmissione, copino file da un processo all'altro. In particolare, il processo servente attende su un socket una connessione, legge un nomefile e successivamente il contenuto del file da salvare con quel nomefile nella directory corrente. Il cliente apre un file (il cui nome sia passato come argomento della riga di comando) e lo copia sul socket connesso con il servente.

Si realizzino due versioni dei programmi:

- una che utilizza i socket `AF_INET` con TCP (protocollo di default di questi socket quando sono aperti con modalità `SOCK_STREAM`)
- una che utilizza i socket `AF_UNIX`, sempre con TCP.

Suggerimenti:

Si faccia in modo che il processo servente legga il nome del file da salvare utilizzando come carattere di fine nomefile il CR. All'accettazione di una nuova connessione, si legga in un `char nomefile[MAXFILENAME]` un carattere alla volta dal socket, fino a quando il carattere letto non sia un CR. Successivamente si usi un altro buffer (magari più grande) per copiare il contenuto del file dal socket al disco

```
s = socket (...);  
  
connect (...);  
  
fd = open (nomefile, ...);  
write (s, nomefile, ...);  
while ((n=read (fd, ...)) > 0)  
    write (s, ...);  
close (s);  
close (fd);
```

Cliente

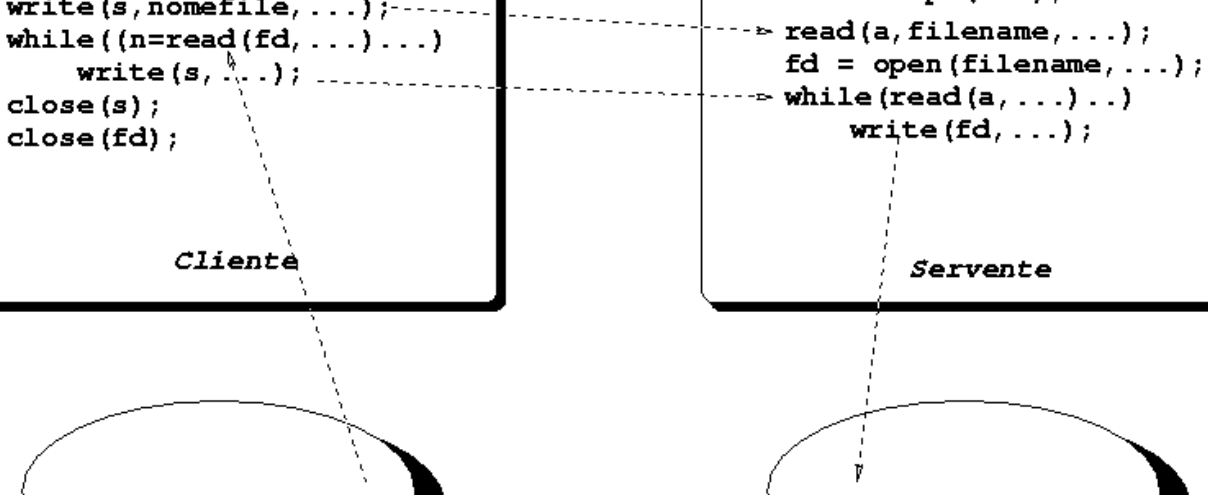
```
s = socket (...);  
bind (...);  
listen (...);  
  
while (true)  
    a = accept (...);  
    > read (a, filename, ...);  
    fd = open (filename, ...);  
    > while (read (a, ...) > 0)  
        write (fd, ...);
```

Servente



Cliente

Servente



Soluzione Esercizio 1

Codice del processo che riceve un messaggio e lo rispedisce al mittente

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <errno.h>

#define MAXBUF 1024

int main(int argc, char * argv[])
{
    int server_socket,connect_socket;
    unsigned int client_addr_len;
    int retcode,portno;
    struct sockaddr_in client_addr, server_addr;
    char line[MAXBUF],hostname[MAXBUF];

    if(argc==1) {
        printf("Usage: \n%s portno\n", argv[0]);
        return(0);
    }
    portno = atoi(argv[1]);
    printf("sponda: fase di inizializzazione\n");
    server_socket = socket(AF_INET,SOCK_STREAM,0);
    if(server_socket == -1) { perror("aprendo il socket del server"); return(-1);}

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(portno);
    gethostname(hostname,MAXBUF);
    memcpy(&server_addr.sin_addr,(gethostbyname(hostname)->h_addr),
        sizeof(server_addr.sin_addr));
    retcode = bind(server_socket, (struct sockaddr *)&server_addr,
        sizeof(server_addr));
    if(retcode == -1) { perror("pubblicando il socket");return(-1); }
    listen(server_socket,1);
    printf("Server: si entra nel ciclo che accetta le connessioni\n");
    client_addr_len = sizeof(client_addr);
    while((connect_socket =
        accept(server_socket,
            (struct sockaddr *) & client_addr,
            &client_addr_len)) != -1) {
#ifdef DEBUG
        printf("Server: accettata nuova connessione\n");
#endif
        retcode = read(connect_socket,line,MAXBUF);
        if(retcode == -1) {
            perror("READING");
            return(errno);
        }
        write(connect_socket,line,retcode);
        close(connect_socket);
#ifdef DEBUG
        printf("Server: fine connessione\n");
#endif
    }
}
```

```

}

printf("Chiusura dei lavori ... \n");
close(server_socket);
return(0);
}

```

Codice del processo che spedisce n messaggi alla sponda, aprendo di volta in volta una nuova connessione

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <errno.h>
#include <time.h>
#include <sys/time.h>

#include "timeutil.h"

int main(int argc, char * argv[])
{
    int client_socket;
    int i,retcode,portno,volte;
    struct sockaddr_in server_addr;
    char * hostname, msg;
    struct timeval start,stop,*diff;

    if(argc==1){
        printf("Usage:\n%s hostsponda porta volte\n",argv[0]);
        return(0);
    }
    portno = atoi(argv[2]);
    volte = atoi(argv[3]);
    hostname = argv[1];

    printf("pinger (%d): fase di inizializzazione (host %s porta %d volte %d\n",
        getpid(),hostname,portno,volte);

    gettimeofday(&start,NULL); // la prima e' per la cache
    gettimeofday(&start,NULL);
    for(i=0;i<volte;i++) {
        msg='a'+(i%26);
        client_socket = socket(AF_INET,SOCK_STREAM,0);
        if(client_socket == -1) {
            perror("aprendo il socket del cliente"); return(-1);}

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(portno);
        memcpy(&server_addr.sin_addr,(gethostbyname(hostname)->h_addr),
            sizeof(server_addr.sin_addr));
        retcode = connect(client_socket,
            (struct sockaddr *)&server_addr,
            sizeof(server_addr));
        if(retcode == -1) { perror("connettendo il socket");return(errno); }
        retcode = write(client_socket,&msg,sizeof(char));
        if(retcode == -1) { perror("scrivendo il messaggio"); return(errno); }
    }
}

```

```

    retcode = read(client_socket,&msg,sizeof(char));
    if(retcode == -1) {perror("leggendo il pong"); return(errno);}
    close(client_socket);
}
gettimeofday(&stop,NULL);
diff = timeval_diff(start,stop);
printf("Elapsed time %ld sec %ld usec \n",
        diff->tv_sec, diff->tv_usec);
printf("Single connection takes (average) %f secs\n",
        timediff2secs(*diff) / ((float) volte));
return(0);
}

```

Codice del processo che spedisce n messaggi alla sponda, aprendo una sola volta una nuova connessione e ciclando solo su letture/scritture

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <errno.h>
#include <time.h>
#include <sys/time.h>

#include "timeutil.h"

int main(int argc, char * argv[])
{
    int client_socket;
    int i,retcode,portno,volte,dimmsg;
    struct sockaddr_in server_addr;
    char * hostname, * msg;
    struct timeval start,stop,*diff;

    if(argc==1){
        printf("Usage:\n%s hostsponda porta volte [dimensionemsg]\n",argv[0]);
        return(0);
    }
    portno = atoi(argv[2]);
    volte = atoi(argv[3]);
    hostname = argv[1];
    if(argc==5)
        dimmsg = atoi(argv[4]);
    else
        dimmsg = 1;
    msg = calloc(sizeof(char),dimmsg);
    for(i=0;i<dimmsg;i++)
        msg[i] = 'a';

    printf("pinger (%d): fase di inizializzazione (host %s porta %d volte %d\n",
           getpid(),hostname,portno,volte);

    client_socket = socket(AF_INET,SOCK_STREAM,0);
    if(client_socket == -1) {
        perror("aprendo il socket del cliente"); return(-1);}

    server_addr.sin_family = AF_INET;

```

```

server_addr.sin_port = htons(portno);
memcpy(&server_addr.sin_addr, (gethostbyname(hostname)->h_addr),
       sizeof(server_addr.sin_addr));
retcode = connect(client_socket,
                  (struct sockaddr *)&server_addr,
                  sizeof(server_addr));
if(retcode == -1) { perror("connettendo il socket");return(errno); }

gettimeofday(&start,NULL); // la prima e' per la cache
gettimeofday(&start,NULL);
for(i=0;i<volte;i++) {
    retcode = write(client_socket,msg,sizeof(char)*dimmsg);
    if(retcode == -1) { perror("scrivendo il messaggio"); return(errno); }
    retcode = read(client_socket,msg,sizeof(char)*dimmsg);
    if(retcode == -1) {perror("leggendo il pong"); return(errno);}
}
gettimeofday(&stop,NULL);
close(client_socket);

diff = timeval_diff(start,stop);
printf("Elapsed time is %ld secs %ld usecs\n", diff->tv_sec, diff->tv_usec);
printf("Single RTT takes (average, %d bytes messages) %f\n",
       dimmsg, timediff2secs(*diff) / ((float) volte));
return(0);
}

```

delle utility per il calcolo dei tempi con la gettimeofday

```

#include <sys/time.h>
//#include <limits.h>
#include <stdlib.h>
#ifdef DEBUG
#include <stdio.h>
#endif

/* computes the difference between two timeval s */

struct timeval * timeval_diff(struct timeval start, struct timeval stop)
{
    struct timeval * new = calloc(sizeof(struct timeval),1);

#ifdef DEBUG
    printf("%10ld\t%10ld\n%10ld\t%10ld\n",
           start.tv_sec, start.tv_usec,
           stop.tv_sec, stop.tv_usec);
#endif
    new->tv_sec = (start.tv_sec > stop.tv_sec ?
                  1000000L - start.tv_sec + stop.tv_sec :
                  stop.tv_sec - start.tv_sec);
    new->tv_usec = (start.tv_usec > stop.tv_usec ?
                   1000000L - start.tv_usec + stop.tv_usec :
                   stop.tv_usec - start.tv_usec);

#ifdef DEBUG
    printf("%10ld\t%10ld\n", new->tv_sec, new->tv_usec);
#endif

    return(new);
}

float timediff2secs(struct timeval t)
{
    float d;

```

```
d =
    (((float) t.tv_usec) / (1000000.0)) +
    ((float) t.tv_sec);

return(d);
}
```

Relativo file header

```
/* computes the difference between two timeval struct */
struct timeval * timeval_diff(struct timeval start, struct timeval stop);
/* converts a timeval difference into float seconds */
float timediff2secs(struct timeval t);
```

Soluzione Esercizio 3

```
//
// produttore consumatore
// un thread produce strutture (stringa + intero)
// un thread le consuma
// interazione mediante var condivisa e semafori
//

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <errno.h>
#include <string.h>
#include <semaphore.h>

// tempo massimo di produzione
#define MAXSEC 5

// tipo dei messaggi
#define MAXC 128
typedef struct __messaggio {
    int intero;
    char stringa[MAXC];
} MESSAGGIO;

// buffer condiviso
MESSAGGIO buffer;

// semaforo per bufferpieno
sem_t pieno;
// semaforo per buffer vuoto
sem_t vuoto;

void * produttore(int *);
void * consumatore(int *);

int main(int argc, char * argv[])
{
    int volte, ret;
    pthread_t tid_consumatore, tid_prodotto;

    if(argc==2)
        volte = atoi(argv[1]);
    else
        volte = 10;

    sem_init(&pieno,0,0);
    sem_init(&vuoto,0,1);

    ret = pthread_create(&tid_consumatore,NULL,
                        (void (*)(void*))consumatore,&volte);
    if(ret != 0) {
        perror("Creando il thread del consumatore");
        return(errno);
    }
    ret = pthread_create(&tid_prodotto,NULL,
                        (void (*)(void*))produttore,&volte);
    if(ret != 0) {
        perror("Creando il thread del consumatore");
        return(errno);
    }
}
```

```

}

// attesa della terminazione dei thread
ret = pthread_join(tid_prodotto, NULL);
if (ret != 0) {
    perror("attendendo la terminazione del thread produttore");
    return(errno);
}
ret = pthread_join(tid_consumatore, NULL);
if (ret != 0) {
    perror("attendendo la terminazione del thread consumatore");
    return(errno);
}

return(0);
}

void * consumatore (int * volte) {
    int i;

    pause();
    for(i=0; i<*volte; i++) {
        sem_wait(&pieno);
        printf("Consumatore consuma <%d,%s> \n",
            buffer.intero, buffer.stringa);
        sem_post(&vuoto);
    }
    return(NULL);
}

void * produttore (int * volte) {
    int i, secondi;

    srand(getpid());

    pause();
    for(i=0; i<*volte; i++) {
        sem_wait(&vuoto);
        printf("Produttore: inizio produzione\n");
        buffer.intero = i;
        sprintf(buffer.stringa, "Iterazione %d", i);
        secondi = rand() % MAXSEC;
        sleep(secondi);
        printf("Produttore produce <%d,%s> \n",
            buffer.intero, buffer.stringa);
        sem_post(&pieno);
    }
    return(NULL);
}

```

Soluzione Esercizio 2

Server con variabili condizione

```
#include <stdio.h>
#include <pthread.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "MultiServer.h"

ITEM * list = NULL;
int shared = 0;

void * conn_handler (int *);
void print_operations(void);
void logger(char *);

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t changes = PTHREAD_COND_INITIALIZER;

int main(int argc, char * argv[])
{
    int sock, port;
    struct sockaddr_in addr;
    char hostname[MAXHOSTNAME];
    struct hostent * hent;
    pthread_t printer, logger;
    char * filelog;

    if(argc < 2) {
        printf("Usage is:\n%s portno [logfile]\n", argv[0]); return(0);
    }

    port = atoi(argv[1]);

    if(argc>2)
        filelog = argv[2];
    else
        filelog = NULL;

    sock = socket(PF_INET, SOCK_STREAM, 0);
    if(sock == -1) {
        perror("Creating the server socket"); exit(errno);
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    gethostname(hostname, MAXHOSTNAME);
    hent = gethostbyname(hostname);
    if(hent == NULL) {
        perror("Finding out host address"); return(errno);
    }
    addr.sin_addr = *((struct in_addr *) hent->h_addr);
```

```

if(bind(sock, (struct sockaddr *)&addr, sizeof(addr))== -1) {
    perror("Binding socket"); return(errno);
}

if(listen(sock,5) == -1) {
    perror("listening"); return(errno);
}

if(pthread_create(&printer,NULL,
                (void * (*)(void *))print_operations,NULL)!=0) {
    perror("creating printer thread");
    return(errno);
}
pthread_detach(printer);

if(pthread_create(&loggert,NULL,
                (void * (*)(void *))logger,
                (void *)filelog)!=0) {
    perror("creating logger thread");
    return(errno);
}
pthread_detach(loggert);

while(1==1) {
    struct sockaddr_in from_addr;
    int retcode;
    int from_addr_len = sizeof(struct sockaddr_in);
    int * acc_sock = calloc(sizeof(int),1);
    pthread_t * new_tid = calloc(sizeof(pthread_t),1);

    *acc_sock = accept(sock, (struct sockaddr *)&from_addr, &from_addr_len);
    retcode = pthread_create(new_tid,
                            NULL,
                            (void * (*)(void *)) conn_handler,
                            (void *) acc_sock);

    if(retcode != 0) {
        perror("creating thread"); return(errno);
    }
    pthread_detach(*new_tid); // we'll not wait for completion
    free(new_tid);
    continue;
}

return(0);
}

void * conn_handler (int * acc_sock) {
    int sock = *acc_sock;
    int retcode, looping = (1==1);
    MESSAGE msg;

    printf("Thread %ld started\n",pthread_self());
    do {
        // reading ...
        retcode = read(sock, &msg, sizeof(msg));
        if(retcode == -1) {
            looping = (1==0);
        } else {
            if(retcode == 0) {
                // client closed connection, exit
                looping = (1==0);
            }
        }
    }
}

```

```

    } else {
        if(retcode != sizeof(msg)) {
            printf("error reading\n");
            return(NULL);
        }
        // insert into list
        pthread_mutex_lock(&lock);    // critical section start
        {
            // remember operations
            ITEM * new = calloc(sizeof(ITEM),1);
            ITEM * temp = list;

            strcpy(new->hostname,msg.hostname);
            new->pid = msg.pid;
            new->incr = msg.incr;
            new->tid = pthread_self();

            list = new;                // LIFO
            new->next = temp;

            shared += msg.incr;        // do operation
        }
        // signal logger
        pthread_cond_signal(&changes);
        pthread_mutex_unlock(&lock); // critical section end
    }
} while(looping);

printf("Thread %ld exiting\n",pthread_self());
pthread_exit(NULL);
}

void print_operations(void) {
    ITEM * p;

    printf("Printing thread started (%ld)\n", pthread_self());
    while(1==1) {
        // printf("Printing thread waiting (%ld)\n", pthread_self());
        sleep(10);                // legal !!!
        // printf("Printing thread printing (%ld)\n", pthread_self());
        pthread_mutex_lock(&lock);
        p = list;
        while(p!=NULL) {
            printf("*** printer thread --> %s (%d %d) op = %d\n",
                p->hostname,p->pid,p->tid,p->incr);
            p = p->next;
        }
        pthread_mutex_unlock(&lock);
    }
    pthread_exit(NULL);
}

void logger(char * file) {
    int fd;

    printf("Thread logger started (%ld)\n",pthread_self());
    if(file == NULL) {
        fd = fileno(stdin);
    } else {
        fd = open(file,O_WRONLY|O_CREAT|O_TRUNC,0644);
        if(fd == -1) {
            perror("creating log file");
            pthread_exit(NULL);
        }
    }
}

```

```

    }
}

while(1==1) {
    char buffer[128];

    pthread_mutex_lock(&lock);
    pthread_cond_wait(&changes, &lock);
    sprintf(buffer,"shared int = %d\n",shared);
    pthread_mutex_unlock(&lock);
    write(fd,buffer,strlen(buffer));
}
}

```

relativo header file

```

#define MAXNAME 256
#define MAXHOSTNAME 1024

typedef struct __item {
    char hostname[MAXNAME];
    pid_t pid;
    pthread_t tid;
    int incr;
    struct __item * next;
} ITEM;

typedef struct {
    int incr;
    char hostname[MAXHOSTNAME];
    int pid;
} MESSAGE;

typedef struct {
    int val;
} ACK;

```

Server con semafori

```

#include <stdio.h>
#include <pthread.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <semaphore.h>

#include "MultiServer.h"

ITEM * list = NULL;
int shared = 0;

void * conn_handler (int *);
void print_operations(void);
void logger(char *);

```

```

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
sem_t ready;

int main(int argc, char * argv[])
{
    int sock, port;
    struct sockaddr_in addr;
    char hostname[MAXHOSTNAME];
    struct hostent * hent;
    pthread_t printer, loggert;
    char * filelog;

    if(argc < 2) {
        printf("Usage is:\n%s portno [logfile]\n", argv[0]); return(0);
    }

    port = atoi(argv[1]);

    if(argc>2)
        filelog = argv[2];
    else
        filelog = NULL;

    sock = socket(PF_INET, SOCK_STREAM, 0);
    if(sock == -1) {
        perror("Creating the server socket"); exit(errno);
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    gethostname(hostname, MAXHOSTNAME);
    hent = gethostbyname(hostname);
    if(hent == NULL) {
        perror("Finding out host address"); return(errno);
    }
    addr.sin_addr = *((struct in_addr *) hent->h_addr);

    if(bind(sock, (struct sockaddr *)&addr, sizeof(addr))== -1) {
        perror("Binding socket"); return(errno);
    }

    if(listen(sock, 5) == -1) {
        perror("listening"); return(errno);
    }

    sem_init(&ready, 0, 0);

    if(pthread_create(&printer, NULL,
                    (void * (*)(void *))print_operations, NULL) != 0) {
        perror("creating printer thread");
        return(errno);
    }
    pthread_detach(printer);

    if(pthread_create(&loggert, NULL,
                    (void * (*)(void *))logger,
                    (void *)filelog) != 0) {
        perror("creating logger thread");
        return(errno);
    }
    pthread_detach(loggert);

    while(1==1) {

```

```

struct sockaddr_in from_addr;
int retcode;
int from_addr_len = sizeof(struct sockaddr_in);
int * acc_sock = calloc(sizeof(int),1);
pthread_t * new_tid = calloc(sizeof(pthread_t),1);

*acc_sock = accept(sock, (struct sockaddr *)&from_addr, &from_addr_len);
retcode = pthread_create(new_tid,
                        NULL,
                        (void * (*)(void *)) conn_handler,
                        (void *) acc_sock);

if(retcode != 0) {
    perror("creating thread"); return(errno);
}
pthread_detach(*new_tid); // we'll not wait for completion
free(new_tid);
continue;
}

return(0);

}

void * conn_handler (int * acc_sock) {
    int sock = *acc_sock;
    int retcode, looping = (1==1);
    MESSAGE msg;

    printf("Thread %ld started\n",pthread_self());
    do {
        // reading ...
        retcode = read(sock, &msg, sizeof(msg));
        if(retcode == -1) {
            looping = (1==0);
        } else {
            if(retcode == 0) {
                // client closed connection, exit
                looping = (1==0);
            } else {
                if(retcode != sizeof(msg)) {
                    printf("error reading\n");
                    return(NULL);
                }
                // insert into list
                pthread_mutex_lock(&lock); // critical section start
                { // remember operations
                    ITEM * new = calloc(sizeof(ITEM),1);
                    ITEM * temp = list;

                    strcpy(new->hostname,msg.hostname);
                    new->pid = msg.pid;
                    new->incr = msg.incr;
                    new->tid = pthread_self();

                    list = new; // LIFO
                    new->next = temp;

                    shared += msg.incr; // do operation
                }
                pthread_mutex_unlock(&lock); // critical section end
                // signal logger
                sem_post(&ready);
            }
        }
    }
}

```

```

    }
} while(looping);

printf("Thread %ld exiting\n",pthread_self());
pthread_exit(NULL);
}

void print_operations(void) {
    ITEM * p;

    printf("Printing thread started (%ld)\n", pthread_self());
    while(1==1) {
        // printf("Printing thread waiting (%ld)\n", pthread_self());
        sleep(10); // legal !!!
        // printf("Printing thread printing (%ld)\n", pthread_self());
        pthread_mutex_lock(&lock);
        p = list;
        while(p!=NULL) {
            printf("*** printer thread --> %s (%d %d) op = %d\n", p->hostname,p-
>pid,p->tid,p->incr);
            p = p->next;
        }
        pthread_mutex_unlock(&lock);
    }
    pthread_exit(NULL);
}

void logger(char * file) {
    int fd;

    printf("Thread logger started (%ld)\n",pthread_self());
    if(file == NULL) {
        fd = fileno(stdin);
    } else {
        fd = open(file,O_WRONLY|O_CREAT|O_TRUNC,0644);
        if(fd == -1) {
            perror("creating log file");
            pthread_exit(NULL);
        }
    }
}

while(1==1) {
    char buffer[128];

    sem_wait(&ready);

    pthread_mutex_lock(&lock); // critical section access
    sprintf(buffer,"shared int = %d\n",shared);
    pthread_mutex_unlock(&lock);
    write(fd,buffer,strlen(buffer));
}
}

```

relativo header file

```

#define MAXNAME 256
#define MAXHOSTNAME 1024

typedef struct __item {
    char hostname[MAXNAME];
    pid_t pid;
    pthread_t tid;
    int incr;
}

```

```

    struct __item * next;
} ITEM;

typedef struct {
    int incr;
    char hostname[MAXHOSTNAME];
    int pid;
} MESSAGE;

typedef struct {
    int val;
} ACK;

```

Client

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>

#include "MultiServer.h"

int main(int argc, char * argv[])
{
    int portno,operation,sock;
    char * server;
    MESSAGE msg;
    struct sockaddr_in addr;
    struct hostent * hent;

    // parameters: client servername serverport operation
    if(argc==1) {
        printf("Usage is:\n%s servername serverport operation\n",argv[0]);
        return(0);
    }
    server = argv[1];
    portno = atoi(argv[2]);
    operation = atoi(argv[3]);

    // preparing message
    msg.pid = getpid();
    gethostname(msg.hostname,MAXHOSTNAME);
    msg.incr = operation;

    // preparing connection
    if((sock = socket(PF_INET,SOCK_STREAM,0)) == -1) {
        perror("Creating socket");
        return(errno);
    }
    hent = gethostbyname(server);
    if(hent == NULL) {
        perror("getting server address");
        return(errno);
    }
    addr.sin_family = AF_INET;
    addr.sin_port = htons(portno);
    addr.sin_addr = *((struct in_addr *) hent->h_addr);

```

```
if(connect(sock,(struct sockaddr *)&addr, sizeof(addr)) == -1) {
    perror("connecting");
    return(errno);
}

// sending message
if(write(sock,&msg,sizeof(msg)) == -1) {
    perror("writing message");
    return(errno);
}

// closing ...
close(sock);

// end work
return(0);
}
```

Soluzione Esercizio 4

Soluzione 1: Cliente e server che realizzano la copia di 1 file prendendo il nome del file da copiare (o da scrivere) come parametro della riga di comando

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/in.h>
#include <netdb.h>
#include <signal.h>
#include <fcntl.h>

#define MAXBUF      8192
#define MAX         128
#define SERVERPORT 3001

int main(int argc, char * argv[])
{
    int client_socket,fd;
    int retcode,letti;
    struct sockaddr_in  server_addr;
    char message[MAXBUF];
    char *nomehost,*filename;

    if(argc==1) {
        printf("Usage:\n%s nomefile nomehost\n",argv[0]);
        return(0);
    }
    filename = argv[1];
    nomehost = argv[2];
    printf("Client (%d): fase di inizializzazione\n",getpid());
    client_socket = socket(AF_INET,SOCK_STREAM,0);
    if(client_socket == -1) {
        perror("aprendo il socket del cliente"); return(-1);}

    server_addr.sin_family = AF_INET;
    server_addr.sin_port   = htons(SERVERPORT);
    memcpy(&server_addr.sin_addr,(gethostbyname(nomehost)->h_addr),
        sizeof(server_addr.sin_addr));
    retcode = connect(client_socket,
        (struct sockaddr *)&server_addr,
        sizeof(server_addr));
    if(retcode == -1) { perror("connettendo il socket");return(-1); }
    fd = open(filename,O_RDONLY);
    if(fd == -1) {
        perror("aprendo il file");
        return(-3);
    }
    do {
        letti = read(fd,message,MAXBUF);
        if(letti > 0) { /* solo se la lettura ha avuto buon fine */
            retcode = write(client_socket,message,letti);
            if(retcode == -1) {
                perror("scrivendo il messaggio");
                return(-3);
            }
        }
    }
    while (letti > 0);
}
```

```

    close(fd);
    close(client_socket);
    return(0);
}

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/in.h>
#include <netdb.h>
#include <signal.h>

#define MAXBUF      8192
#define SERVERPORT 3001

int main(int argc, char * argv[])
{
    int server_socket,connect_socket;
    unsigned int client_addr_len;
    int retcode,fd;
    struct sockaddr_in client_addr, server_addr;
    char line[MAXBUF],hostname[MAXBUF];

    if(argc==1) {
        printf("Usage:\n%s nomefilelocale\n",argv[0]);
        return(00);
    }
    printf("Server: fase di inizializzazione\n");
    server_socket = socket(AF_INET,SOCK_STREAM,0);
    if(server_socket == -1) {
        perror("aprendo il socket del server");
        return(-1);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port   = htons(SERVERPORT);
    if(gethostname(hostname,MAXBUF) == -1) {
        perror("cercando il nome dell'host");
        return(-1);
    }
    memcpy(&server_addr.sin_addr,(gethostbyname(hostname)->h_addr),
        sizeof(server_addr.sin_addr));
    retcode = bind(server_socket,
        (struct sockaddr *)&server_addr,
        sizeof(server_addr));
    if(retcode == -1) { perror("pubblicando il socket");return(-1); }
    listen(server_socket,1);
    printf("Server: attendo connessione\n");
    client_addr_len = sizeof(client_addr);
    connect_socket = accept(server_socket,
        (struct sockaddr *) & client_addr,
        &client_addr_len);
    printf("Server: accettata nuova connessione\nApro file locale %s",argv[1]);
    fd = open(argv[1],O_WRONLY|O_TRUNC|O_CREAT,0644);
    if(fd == -1) {
        perror("aprendo il file locale");
        return(-2);
    }
    do {

```

```

    retcode = read(connect_socket,line,MAXBUF);
    if(retcode != -1)
        write(fd,line,retcode);
} while(retcode > 0);
close(fd);
printf("\nFine del messaggio, chiusura della connessione\n");
close(connect_socket);

printf("Chiusura dei lavori ... \n");
close(server_socket);
return(0);
}

```

Soluzione 2: in questo caso il lettore del file comunica come prima informazione il nome del file allo scrittore. Inoltre lo scrittore attende un numero infinito di connessioni (protocollo buco nero ...)

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/in.h>
#include <netdb.h>
#include <signal.h>
#include <fcntl.h>

/* attenzione: client e server usano buffer di dimensione diversa */
#define MAXBUF    1024

#include "Protocollo.h"

int main(int argc, char * argv[])
{
    int client_socket,fd;
    int retcode,letti;
    struct sockaddr_in server_addr;
    char message[MAXBUF];
    char *nomehost,*filename;
    HEADER hdr;

    /* trattamento dei parametri */
    if(argc==1) {
        printf("Usage:\n%s nomefile nomehost\n",argv[0]);
        return(0);
    }
    filename = argv[1];
    nomehost = argv[2];
    /* preparazione dell'header da mandare al servernte */
    hdr.len = strlen(filename);
    strcpy(hdr.filename,filename);
    /* inizio lavori: apertura socket */
    printf("Client (%d): fase di inizializzazione\n",getpid());
    client_socket = socket(AF_INET,SOCK_STREAM,0);
    if(client_socket == -1) {
        perror("aprendo il socket del cliente"); return(-1);}
    /* preparazione indirizzo socket servernte */
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVERPORT);
    memcpy(&server_addr.sin_addr,(gethostbyname(nomehost)->h_addr),
        sizeof(server_addr.sin_addr));
    /* connessione al servernte */

```

```

retcode = connect(client_socket,
                  (struct sockaddr *)&server_addr,
                  sizeof(server_addr));
if(retcode == -1) { perror("connettendo il socket");return(-1); }
/* invio dell'intestazione con il nome del file da scrivere */
retcode = write(client_socket, (char *)&hdr, sizeof(hdr));
if(retcode == -1) {
    perror("inviando il nome del file al servente");
    return(-5);
}
/* apertura file da trasferire */
fd = open(filename, O_RDONLY);
if(fd == -1) {
    perror("aprendo il file");
    return(-3);
}
/* ciclo di copia del file */
while((letti = read(fd, message, MAXBUF)) > 0) {
    retcode = write(client_socket, message, letti);
    if(retcode == -1) {
        perror("scrivendo il messaggio");
        return(-3);
    }
}
/* chiusura lavori */
close(fd);
close(client_socket);
printf("File %s trasmesso\n", filename);
return(0);
}

```

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/in.h>
#include <netdb.h>
#include <signal.h>

```

```
#define MAXBUF      8192
```

```
#include "Protocollo.h"
```

```

int main(int argc, char * argv[])
{
    int server_socket, connect_socket;
    unsigned int client_addr_len;
    int retcode, fd;
    struct sockaddr_in client_addr, server_addr;
    char line[MAXBUF], hostname[MAXBUF];

    /* creazione del socket */
    printf("Server: fase di inizializzazione\n");
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if(server_socket == -1) {
        perror("aprendo il socket del server");
        return(-1);
    }
    /* pubblicazione dell'indirizzo */

```

```

server_addr.sin_family = AF_INET;
server_addr.sin_port   = htons(SERVERPORT);
if(gethostname(hostname,MAXBUF) == -1) {
    perror("cercando il nome dell'host");
    return(-1);
}
memcpy(&server_addr.sin_addr,(gethostbyname(hostname)->h_addr),
        sizeof(server_addr.sin_addr));
retcode = bind(server_socket,
                (struct sockaddr *)&server_addr,
                sizeof(server_addr));
if(retcode == -1) { perror("pubblicando il socket");return(-1); }
/* dichiarazione del backlog */
listen(server_socket,1);
/* ciclo di accettazione delle connessioni */
printf("Server: attendo connessione\n");
client_addr_len = sizeof(client_addr);
while((connect_socket =
        accept(server_socket,
                (struct sockaddr *) & client_addr,
                &client_addr_len))!= -1) {
    HEADER hdr;

    printf("Server: accettata nuova connessione\nApro file locale %s",argv[1]);
    /* mi procuro il nome del file da scrivere */
    retcode = read(connect_socket,(char *)&hdr,sizeof(hdr));
    if(retcode == -1) {
        perror("leggendo il nome del file da scrivere");
        close(connect_socket);
        break;
    }
    /* apertura del file da scrivere */
    fd = open(hdr.filename,O_WRONLY|O_TRUNC|O_CREAT,0644);
    if(fd == -1) {
        perror("aprendo il file locale");
        return(-2);
    }
    /* ciclo di copia del file sul disco locale */
    while((retcode = read(connect_socket,line,MAXBUF)) > 0) {
        retcode = write(fd,line,retcode);
        if(retcode == -1) {
            perror("scrivendo sul file locale");
            return(-6);
        }
    }
    /* trattamento della terminazione della connessione */
    close(fd);
    printf("\nFine del messaggio, chiusura della connessione\n");
    close(connect_socket);
}
/* fine lavori */
printf("Chiusura dei lavori ... \n");
close(server_socket);
return(0);
}

#define MAXFILENAME 1024

typedef struct __header {
    int len;
    char filename[MAXFILENAME];
} HEADER;

```

```
#define SERVERPORT 3001
```

Soluzione 3: versione del programma che funziona col telnet: prima mando il filename seguito da un CR seguito dal file In questo modo posso usare il telnet per collegarmi al servizio ... invece del Lettore ;-)

In questo caso, lanciare il servente (Scrittore-ASCII) in una finestra Sull'altra finestra

```
telnet hostname 3002
```

```
nomefile CR
```

```
file contents
```

```
Esc Cntl-]
```

```
close
```

per terminare il telnet ... e voila'

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/in.h>
#include <netdb.h>
#include <signal.h>
#include <fcntl.h>

/* attenzione: client e server usano buffer di dimensione diversa */
#define MAXBUF 1024

#include "Protocollo-ASCII.h"

int main(int argc, char * argv[])
{
    int client_socket,fd;
    int retcode,letti;
    struct sockaddr_in server_addr;
    char message[MAXBUF];
    char *nomehost,*filename;

    /* trattamento dei parametri */
    if(argc==1) {
        printf("Usage:\n%s nomefile nomehost\n",argv[0]);
        return(0);
    }
    filename = argv[1];
    nomehost = argv[2];
    /* preparazione dell'header da mandare al servente */
    sprintf(message,"%s\n",filename);
    /* inizio lavori: apertura socket */
    printf("Client (%d): fase di inizializzazione\n",getpid());
    client_socket = socket(AF_INET,SOCK_STREAM,0);
    if(client_socket == -1) {
        perror("aprendo il socket del cliente"); return(-1);}
    /* preparazione indirizzo socket servente */
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVERPORT);
    memcpy(&server_addr.sin_addr,(gethostbyname(nomehost)->h_addr),
        sizeof(server_addr.sin_addr));
    /* connessione al servente */
    retcode = connect(client_socket,
```

```

                (struct sockaddr *)&server_addr,
                sizeof(server_addr));
if(retcode == -1) { perror("connettendo il socket");return(-1); }
/* invio dell'intestazione con il nome del file da scrivere */
retcode = write(client_socket,message,strlen(message));
if(retcode == -1) {
    perror("inviando il nome del file al servente");
    return(-5);
}
/* apertura file da trasferire */
fd = open(filename,O_RDONLY);
if(fd == -1) {
    perror("aprendo il file");
    return(-3);
}
/* ciclo di copia del file */
while((letti = read(fd,message,MAXBUF)) > 0) {
    retcode = write(client_socket,message,letti);
    if(retcode == -1) {
        perror("scrivendo il messaggio");
        return(-3);
    }
}
/* chiusura lavori */
close(fd);
close(client_socket);
printf("File %s trasmesso\n", filename);
return(0);
}

```

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/in.h>
#include <netdb.h>
#include <signal.h>

```

```
#define MAXBUF      8192
```

```
#include "Protocollo-ASCII.h"
```

```

int main(int argc, char * argv[])
{
    int server_socket,connect_socket;
    unsigned int client_addr_len;
    int retcode,fd;
    struct sockaddr_in client_addr, server_addr;
    char line[MAXBUF],hostname[MAXBUF];

    /* creazione del socket */
    printf("Server: fase di inizializzazione\n");
    server_socket = socket(AF_INET,SOCK_STREAM,0);
    if(server_socket == -1) {
        perror("aprendo il socket del server");
        return(-1);
    }
    /* pubblicazione dell'indirizzo */
    server_addr.sin_family = AF_INET;

```

```

server_addr.sin_port = htons(SERVERPORT);
if(gethostname(hostname,MAXBUF) == -1) {
    perror("cercando il nome dell'host");
    return(-1);
}
memcpy(&server_addr.sin_addr,(gethostbyname(hostname)->h_addr),
    sizeof(server_addr.sin_addr));
retcode = bind(server_socket,
    (struct sockaddr *)&server_addr,
    sizeof(server_addr));
if(retcode == -1) { perror("pubblicando il socket");return(-1); }
/* dichiarazione del backlog */
listen(server_socket,1);
/* ciclo di accettazione delle connessioni */
printf("Server: attendo connessione\n");
client_addr_len = sizeof(client_addr);
while((connect_socket =
    accept(server_socket,
        (struct sockaddr *) & client_addr,
        &client_addr_len))!= -1) {
    char message[MAXBUF];
    int c=0;
    char filename[MAXBUF];

    printf("Server: accettata nuova connessione\nApro file locale %s",filename);
    /* mi procuro il nome del file da scrivere */
    while((read(connect_socket,(char *)&message[c],sizeof(char))!=-1) &&
(message[c]!='\n'))
        c++;

    sscanf(message,"%s\n",filename);
    /* apertura del file da scrivere */
    fd = open(filename,O_WRONLY|O_TRUNC|O_CREAT,0644);
    if(fd == -1) {
        perror("aprendo il file locale");
        return(-2);
    }
    /* ciclo di copia del file sul disco locale */
    while((retcode = read(connect_socket,line,MAXBUF)) > 0) {
        retcode = write(fd,line,retcode);
        if(retcode == -1) {
            perror("scrivendo sul file locale");
            return(-6);
        }
    }
    /* trattamento della terminazione della connessione */
    close(fd);
    printf("\nFine del messaggio, chiusura della connessione\n");
    close(connect_socket);
}
/* fine lavori */
printf("Chiusura dei lavori ... \n");
close(server_socket);
return(0);
}

```

```
define SERVERPORT 3002
```

Soluzione 4: questa volta solo il server, da usare con telnet host 3002 GET\nfilename\n oppure PUT\nfilename\n Alla PUT deve fare seguito il file da inviare (cut and paste col mouse sulla finestra del telnet)

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/in.h>
#include <netdb.h>
#include <signal.h>
#include <sys/stat.h>
#include <errno.h>

#define MAXBUF      8192

#include "ftp.h"

/* esterni perche' da chiudere ... */
int server_socket,connect_socket;

void termina(int);

int main(int argc, char * argv[])
{
    unsigned int client_addr_len;
    int retcode,fd;
    struct sockaddr_in client_addr, server_addr;
    char line[MAXBUF],hostname[MAXBUF];

    /* creazione del socket */
    printf("Server: fase di inizializzazione\n");
    server_socket = socket(AF_INET,SOCK_STREAM,0);
    if(server_socket == -1) {
        perror("aprendo il socket del server");
        return(-1);
    }
    /* accetta ^C terminando il socket */
    signal(SIGINT,termina);
    /* pubblicazione dell'indirizzo */
    server_addr.sin_family = AF_INET;
    server_addr.sin_port   = htons(SERVERPORT);
    if(gethostname(hostname,MAXBUF) == -1) {
        perror("cercando il nome dell'host");
        return(-1);
    }
    memcpy(&server_addr.sin_addr,(gethostbyname(hostname)->h_addr),
        sizeof(server_addr.sin_addr));
    retcode = bind(server_socket,
        (struct sockaddr *)&server_addr,
        sizeof(server_addr));
    if(retcode == -1) { perror("pubblicando il socket");return(-1); }
    /* dichiarazione del backlog */
    listen(server_socket,1);
    /* ciclo di accettazione delle connessioni */
    printf("Server: attendo connessione\n");
    client_addr_len = sizeof(client_addr);
    while((connect_socket =
        accept(server_socket,
            (struct sockaddr *) & client_addr,
            &client_addr_len))!= -1) {
        char message[MAXBUF];
        int c=0;
        char filename[MAXBUF];

```

```

printf("Server: accettata nuova connessione\nApro file locale %s",filename);
/* leggo un comando */
while((read(connect_socket,(char *)&message[c],sizeof(char))!=-1)
    && (message[c]!='\n'))
    c++;
/* cerco di capire se e' una GET o una PUT */
c = 0;
if(strncmp(message,"GET",3)==0) {
    struct stat statbuf;
    /* devo spedire un file al cliente: leggo il nome del file */
    while((read(connect_socket,(char *)&message[c],sizeof(char))!=-1)
        && (message[c]!='\n'))
        c++;
    sscanf(message,"%s\n",filename);
    if(stat(filename,&statbuf)== -1 && errno==ENOENT) {
        /* il file non esiste */
        sprintf(message,"%s: no such file\n",filename);
        retcode = write(connect_socket,message,strlen(message));
        if(retcode == -1) {
            perror("scrivendo la risposta");
            return(-7);
        }
    } else {
        int letti;
        /* copio il file sul socket (bidirezionale!) */
        fd = open(filename,O_RDONLY);
        while((letti=read(fd,message,MAXBUF))>0) {
            retcode = write(connect_socket,message,letti);
            if(retcode == -1) {
                perror("inviando il file");
                return(-9);
            }
        }
        close(fd);
    }
}
if(strncmp(message,"PUT",3)==0) {
    c = 0;
    /* devo ricevere un file dal cliente: leggo il nome del file */
    while((read(connect_socket,(char *)&message[c],sizeof(char))!=-1)
        && (message[c]!='\n'))
        c++;
    /* apertura del file da scrivere */
    sscanf(message,"%s\n",filename);
    fd = open(filename,O_WRONLY|O_TRUNC|O_CREAT,0644);
    if(fd == -1) {
        perror("aprendo il file locale");
        return(-2);
    }
    /* ciclo di copia del file sul disco locale */
    while((retcode = read(connect_socket,line,MAXBUF)) > 0) {
        if(strncmp(line,"EOF",3) == 0) {
            /* fine lavori per questo file */
            break;
        }
        retcode = write(fd,line,retcode);
        if(retcode == -1) {
            perror("scrivendo sul file locale");
            return(-6);
        }
    }
}
/* trattamento della terminazione della connessione */

```

```
        close(fd);
    }
    printf("\nFine del messaggio, chiusura della connessione\n");
    close(connect_socket);
}
/* fine lavori */
printf("Chiusura dei lavori ... \n");
close(server_socket);
return(0);
}

void termina(int sig)
{
    close(connect_socket);
    close(server_socket);
    return;
}
```