Applicazione Client-Server con Server Concorrente
Specifiche

Il progetto consiste nello sviluppo di un'applicazione client/server. Client e server devono comunicare tramite socket TCP. . Il server deve essere concorrente e la concorrenza deve essere implementata con i thread POSIX. Il thread main deve rimanere perennemente in attesa di nuove connessioni e le deve smistare ad un pool (insieme) di thread che hanno il compito di gestire le richieste.
L'applicazione da sviluppare, chiamiamola remote-compressor , deve consentire ad un client di mandare file ad un server per poi farsi restituire un archivio (tar) compresso con un algoritmo di compressione scelto dal client, contenente tali file. A tale scopo devono essere realizzati due programmi, centralized-compressor dal lato server e compressor-client dal lato client.

Lato client

Il programma centralized-compressor è un programma interattivo che consente al client:

- la scelta dell'algoritmo di compressione
- la scelta del nome dell'archivio
- la possibilità di visualizzare le configurazioni scelte
- l'invio di uno o più file al server
- la ricezione di un archivio (**tar**) compresso con i file mandati dal client stesso

Il comando per aprire una sessione di lavoro ha la seguente sintassi:

compressor-client <host-remoto> <porta>

dove *host-remoto* è l'indirizzo IP della macchina dove gira il server centralized-compressor e *porta* è la porta su cui è in ascolto il server. Una volta mandato in esecuzione, compressor-client deve stampare su video un messaggio che informa il client dell'avvenuta connessione e successivamente deve far apparire sullo schermo un prompt (esempio, remote-compressor >). Successivamente l'utente può digitare i comandi per interagire con il server. I comandi che possono essere mandati in esecuzione sono i seguenti:

- **`help`**: tale comando deve mostrare a video una breve guida dei comandi disponibili.
- **`configure-compressor`** `[compressor]`: tale comando deve configurare il server in maniera tale da impostare l'algoritmo di compressione scelto. Gli algoritmi di compressione **devono essere almeno 2**:
  - bzip2
  - gnuzip

  Si riserba allo studente l'eventuale inserimento di ulteriori algoritmi (rar, zip ecc). Una volta eseguito il comando il server dovrà rispondere con un messaggio di successo/errore che verrà visualizzato sul client in modo da capire se la configurazione dell'algoritmo di compressione abbia avuto successo o meno.

- **`configure-name`** `[name]:` questo comando deve impostare il nome dell'archivio che vogliamo ricevere. Anche in questo caso il server dovrà rispondere con un messaggio per capire l'esito del comando.
- **`show-configuration:`** tale comando restituisce il nome scelto per l'archivio e il compressore, se tali parametri non sono stati ancora configurati mostra quelli di default.
- **`send`** `[file]`: tale comando prende come parametro il path del file locale che deve essere inviato al server (*local-file*). Per eseguire tale operazione il client deve aprire il file *local-file* e andarne a leggere il contenuto. In caso di errore nell'apertura del file deve essere stampato a video un opportuno messaggio di errore. Una volta letto il file, il client deve inviare al server un comando opportuno fatto seguire dal contenuto del file e dal nome del path dove deve essere inserito il file. Il server prende i dati inviati dal client e li inserisce all'interno del file *remote-file* (mantenere il nome originale). In caso di errore durante la creazione del file, il server deve stampare su video un messaggio di errore e inviare al client un messaggio opportuno. Il client deve ricevere tale messaggio e visualizzarlo. Il file *remote-file* deve essere una copia esatta del file *local-file* (usare l'utility diff per verificare se i due file sono uguali). Nel caso in cui il server non abbia i diritti per eseguire l'operazione richiesta, deve essere spedito al client un messaggio opportuno che lo informa sui problemi incontrati dal server. Questo parametro può essere richiamato n volte per ogni file che vogliamo inserire nell'archivio.
- **`compress`** `[path]:` questo comando crea il targz oppure il tar.bz2 dei file mandati con il comando send e crea il file *compressed-remote-file* con il nome scelto attraverso il comando `configure-name.` Il server dovrà poi aprire tale file, leggerlo, e inviare al client il contenuto. Il server dovrà comunicare al client anche il nome del file. Il client dovrà salvare il file nel *path* specificato. Come per il comando `send` sia server che client dovranno gestire tutte le eccezioni e stampare a video lo stato delle operazioni (successo/errore e tipo). Per esempio se si esegue il comando `compress` senza aver mai eseguito una `send` ovviamente il server dovrà restituire un errore. Se si esegue invece questo comando senza aver eseguito i comandi di `configure-*` il server utilizzerà delle impostazioni di default:
  - **`nome:`** ***archivio***
  - **`compressore:`** ***gnuzip***

  Una volta mandato l'archivio al client il server provvederà ad eliminare tutti i file temporanei.
- **`quit:`** tale comando causa la terminazione della sessione di lavoro iniziata con il comando compressor-client. Più precisamente tale comando chiude il socket con il server ed esce. Il server deve visualizzare un messaggio che attesta la disconnessione del client.

Lato server

Il processo centralized-compressor rappresenta il server del servizio remote-compressor. Tale processo rimane perennemente in ascolto di richieste di connessione provenienti dai client. Quando un client si connette, centralized-compressor deve generare un thread a cui delegare la gestione del servizio offerto e deve tornare ad attendere altre richieste di connessione. Appena un client si connette, il server deve visualizzare un messaggio che indica che il client con indirizzo IP x.x.x.x si è connesso sulla porta yy. La sintassi del comando centralized-compressor è la seguente:

centralized-compressor <porta>
dove *porta* è la porta su cui si deve mettere in ascolto il server. I processi che gestiscono il servizio offerto ai client devono visualizzare tutte le informazioni utili per capire che tipo di operazioni vengono

richieste dai client. Oltre a questo devono anche visualizzare i messaggi di errore di cui abbiamo discusso nei punti precedenti.

Esempio di esecuzione del client:
utente@localhost ~/client $ **./compressor-client 127.0.0.1 4000**
REMOTE COMPRESSOR client, v.0.1
Connected to Server 127.0.0.1 on port 4000
Inserire comandi di lunghezza massima 1024 caratteri!
remote-compressor> **ciao**
CLIENT:comando non valido!
remote-compressor>**help**
I conandi supportati da remote-compressor sono i seguenti:
-) configure-compressor [compressor]
-) configure-name [name]
-) show-configuration
-) send [local-file]
-) compress [path]
-) quit
remote-compressor> **configure-compressor gnuzip**
Compressore configurato correttamente.
remote-compressor> **configure-name archivio_foto**
Nome configurato correttamente.
remote-compressor> **show-configuration**
Nome: archivio_foto
Compressore: gnuzip
remote-compressor>**send foto1.jpg**
File foto1.jpg mandato con successo.
remote-compressor> **send foto2.jpg**
File foto2.jpg mandato con successo.
remote-compressor>**compress .**
Archivio archivo_foto.tar.gz ricevuto con successo.
remote-compressor>**quit**
utente@localhost ~/client $

Esempio di esecuzione del server:
utente@localhost ~/server $ **./centralized-compressor 4000**
porta 4000
REMOTE-COMPRESSOR server, v.0.1
Attesa di connessioni....
CLIENT 127.0.0.1 connesso
CLIENT 127.0.0.1 eseguito comando help
CLIENT 127.0.0.1 eseguito comando configure-compressor gnuzip
CLIENT 127.0.0.1 eseguito comando configure-name archivio_foto
CLIENT 127.0.0.1 eseguito comando show-configuration
SERVER: ricevuto il file foto1.jpg dal client 127.0.0.1
SERVER: ricevuto il file foto2.jpg dal client 127.0.0.1
SERVER compressione in corso archivio_foto.tar.gz richiesta dal client 127.0.0.1
SERVER: spedito file archivio_foto.tar.gz al client 127.0.0.1
SERVER: client 127.0.0.1 chiude la connessione

Note:

- Client e server si scambiano dei dati tramite socket. Prima che inizi ogni scambio è necessario che il ricevente sappia quanti byte deve leggere dal socket. Ad esempio, quando il client invia al server il contenuto del file da copiare, il server non sa a priori quanti bytes deve ricevere. È

quindi necessario che il client invii al server un'informazione che lo metta in condizione di sapere quando ha letto tutti i bytes che il client gli deve inviare.

- Le specifiche non danno vincoli sulla lunghezza dei parametri inseriti dall'utente tramite tastiera. È tuttavia normale che il client imponga un limite su tale lunghezza. In tal caso, È NECESSARIO che tale vincolo sia espressamente comunicato all'utente (ad esempio, tramite la stampa a video di un messaggio di benvenuto).
- NON È AMMESSO CHE VENGANO INVIATI SU SOCKET NUMERI ARBITRARI DI BYTES. Ad esempio, se l'informazione da inviare server (client) è lunga 5 Bytes, DEVONO ESSERE INVIATI SOLAMENTE 5 BYTES (eventualmente 6 se per qualche motivo sensato ha senso mandare anche il carattere di terminazione stringa)

Valutazione del progetto:

Il progetto viene valutato durante lo svolgimento dell'esame. La valutazione prevede le seguenti fasi:

1. Vengono compilati i sorgenti di client e server (la compilazione avviene con l'opzione -Wall, che segnala tutti i warning. Si consiglia vivamente di usare tale opzione anche durante lo sviluppo del progetto, INTERPRETANDO I MESSAGGI DATI DAL COMPILATORE).
2. Viene eseguita l'applicazione per controllarne il funzionamento rispetto alle specifiche fornite.
3. Vengono esaminati i sorgenti per controllare l'implementazione.

```c
// compression-server v.0.1

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>

#define SA struct sockaddr
#define BACKLOG_SIZE 5

#define BZIP2 0
#define GNUZIP 1

/* Argomento da passare al thread */

struct argomenti {
    int sock_cl;
    struct sockaddr_in c_addr;
};

int t_free[BACKLOG_SIZE];
pthread_t t_id[BACKLOG_SIZE];
pthread_t daeid[BACKLOG_SIZE];
```

```
pthread_attr_t attr;


const char *esecuzione[] = {"help","configure-compressor", "configure-name","show-
configuration","send","compress","quit","errore"};
const char *compressore[] = {"bzip2", "gnuzip"};
const char *tar_cond[] = {"-cjf" ,"-czf" };


void* gestisci_richiesta (void* dati){

    /* Strutture dati da usare */

    struct argomenti* argos;
    char cl_paddr [INET_ADDRSTRLEN];
    char messaggio[1024];
    int ret, comando;
    unsigned int mes_len;

    char* buf_file;
    FILE* puntat;
    long fsize;
    struct stat pro;

    int pid;

    int flag;
    int compressor;
    char nome[1024];
    char nome_file[1024];
```

```
char directory[256];
char tar_dir[256];
int vuoto = 1;

strncpy(nome, "archivio", sizeof(nome)-1);
nome[1023]='\0';
flag=1;
compressor = GNUZIP;

/* Creazione cartelle temporanee di lavoro */

argos = (struct argomenti*) dati;
strncpy(directory, "./.temp_", sizeof(directory)-1);
directory[1023]='\0';
sprintf(messaggio,"%d",argos->sock_cl);
strncat(directory, messaggio, sizeof(directory) - strlen(directory) - 1);
messaggio[0]='\0';


strncpy(tar_dir, ".tar_temp_", sizeof(directory)-1);
directory[1023]='\0';
sprintf(messaggio,"%d",argos->sock_cl);
strncat(tar_dir, messaggio, sizeof(tar_dir) - strlen(tar_dir) - 1);
messaggio[0]='\0';

pid = fork();
if (pid == 0){
 execlp("/bin/mkdir", "mkdir", directory,NULL);
}
wait(NULL);
```

```
    pid = fork();
    if (pid == 0){
     execlp("/bin/mkdir", "mkdir", tar_dir,NULL);
    }
    wait(NULL);

    /* Metto in cl_paddr l'indirizzo del client (in formato puntato) */

    inet_ntop(AF_INET, &argos->c_addr.sin_addr, cl_paddr, sizeof(cl_paddr));

    /* Ricevo comando dal client */

    printf("CLIENT %s connesso\n", cl_paddr);
    while(flag){
     ret = recv(argos->sock_cl, &comando, sizeof(comando), MSG_WAITALL);
     switch (comando){
          case 0: /* Comando help */
               strncpy (messaggio, "I comandi supportati da remote-compressor sono i seguenti:\n
-)configure-compressor [compressore]\n -)configure-name [name]\n -)show-configuration\n -)send
[local-file] \n -)compress [path] \n -)quit \n", sizeof(messaggio)-1);
               messaggio[1023] = '\0';
               printf("CLIENT %s: eseguito comando help\n", cl_paddr);
               break;
          case 1: /* Comando configure_compressor */
               ret = recv(argos->sock_cl, &comando, sizeof(comando), MSG_WAITALL);
               if (comando == 2){
                    strncpy (messaggio, "Attenzione compressore non supportato.\nRipetere comando\n",
sizeof(messaggio)-1);
```

```
                    messaggio[1023] = '\0';
               }
               else{
                    compressor = comando;
                    strncpy (messaggio, "Compressore configurato correttamente\n\0",
sizeof(messaggio)-1);
                    messaggio[1023]='\0';

               }

               if(comando == 2)
                    printf ("CLIENT %s: errore nella configurazione del compressore\n", cl_paddr);
               else
                    printf ("CLIENT %s: eseguito comando configure-compressor %s\n",cl_paddr ,
compressore[compressor]);
               break;
          case 2: /* Comando configure_name */
               ret = recv(argos->sock_cl, &mes_len, sizeof(mes_len),MSG_WAITALL);
               if(ret<sizeof(mes_len)){
                    printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
                    exit(1);
               }

               ret = recv(argos->sock_cl, &nome, mes_len, MSG_WAITALL);
               if(ret<mes_len){
                    printf("ERRORE: impossibile inviare il messaggio");
                    exit(1);
               }
               nome[mes_len]='\0';
               strncpy(messaggio, "nome configurato correttamente\n", sizeof(messaggio)-1);
```

```
     messaggio[1023]='\0';
     printf ("CLIENT %s: eseguito comando configure-name %s\n", cl_paddr, nome);
     break;
case 3: /* Comando show-configuration */
     strncpy (messaggio, "Nome: ", sizeof(messaggio)-1);
     messaggio[1023]='\0';
     strncat (messaggio, nome, sizeof(messaggio)-strlen(messaggio)-1);
     strncat (messaggio, "\nCompressore: ", sizeof(messaggio)-strlen(messaggio) - 1);
     strncat (messaggio, compressore[compressor], sizeof(messaggio)-strlen(messaggio)-1);
     strncat (messaggio, " \n", sizeof(messaggio)-strlen(messaggio)-1);
     messaggio[1023]='\0';

     printf ("CLIENT %s: eseguito comando show-configuration\n", cl_paddr);

     break;
case 4: /* Comando send */
     ret = recv(argos->sock_cl, &mes_len, sizeof(mes_len),MSG_WAITALL);
     if(ret<sizeof(mes_len)){
          printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
          exit(1);
     }
     if (mes_len==0){
          strncpy(messaggio, "Comando send abortito\n", sizeof(messaggio));
          messaggio[1023] = '\0';
          break;
     }

     /* Ricezione nome file */

     ret = recv(argos->sock_cl, &nome_file, mes_len, MSG_WAITALL);
```

```
if(ret<mes_len){
     printf("ERRORE: impossibile inviare il messaggio");
     exit(1);
}
nome_file[mes_len]='\0';
ret = recv(argos->sock_cl, &mes_len, sizeof(mes_len),MSG_WAITALL);
if(ret<sizeof(mes_len)){
     printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
     exit(1);
}

/* Ricezione dal socket del file mandato con la send*/

buf_file =(char*)malloc(sizeof(char)*(mes_len + 1));
if (buf_file==NULL){
     printf("Attenzione! Errore nell'allocazione memoria dinamica");
     exit(2);
}

ret = recv(argos->sock_cl, buf_file, mes_len, MSG_WAITALL);
if(ret<mes_len){
     printf("ERRORE: impossibile inviare il messaggio\n");
     exit(1);
}
buf_file[mes_len]='\0';

strncpy(messaggio,directory,sizeof(messaggio)-1);
if(nome_file[0]!='/')
     strncat(messaggio,"/",sizeof(messaggio)-strlen(messaggio)-1);
strncat(messaggio,nome_file,sizeof(messaggio)-strlen(messaggio)-1);
```

```
                /* Scrivo nel filesystem il file passato dal client */

                puntat = fopen(messaggio,"w");
                fsize = (long)mes_len;
                fwrite (buf_file,1,fsize,puntat);
                fclose(puntat);
                free(buf_file);

                vuoto = 0;

                printf("SERVER: ricevuto da %s file %s\n", cl_paddr, nome_file);
                strncpy(messaggio, "File ricevuto correttamente\n", sizeof(messaggio));


                break;
           case 5: /* Comando compress */
                if (vuoto==1){
                     mes_len = 0;
                     ret = send(argos->sock_cl, &mes_len, sizeof(mes_len),0);
                     if(ret<sizeof(mes_len)){
                          printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
                          exit(1);
                     }

                     strncpy(messaggio, "ERRORE: Eseguito comando compress con buffer vuoto. Si prega
di eseguire almeno un comando send prima di riprovare\n", sizeof(messaggio));
                     messaggio[1023]='\0';
                     printf ("CLIENT %s: Eseguito comando compress con buffer vuoto\n", cl_paddr);
                     break;
```

```
                }
                strncpy(nome_file, "../", sizeof(nome_file));
                strncat(nome_file, tar_dir, sizeof(nome_file)-strlen(nome_file)-1);
                strncat(nome_file, "/", sizeof(nome_file)-strlen(nome_file)-1);
                strncat(nome_file, nome, sizeof(nome_file)-strlen(nome_file)-1);
                if(compressor == GNUZIP)
                     strncat(nome_file, ".tar.gz",sizeof(nome_file)-strlen(nome_file)-1);
                else
                     strncat(nome_file, ".tgz", sizeof(nome_file)-strlen(nome_file)-1);
                strncpy(messaggio, "./", sizeof(messaggio));

                /* Cambio directory e eseguo tar */

                chdir(directory);
                pid = fork();
                if (pid == 0){
                     execlp("/usr/bin/tar", "tar", tar_cond[compressor], nome_file, messaggio, NULL);
                     exit(1);
                }
                wait(NULL);

                printf("SERVER: compressione file %s richesta dal client %s...\n", nome_file,
cl_paddr);

                puntat = fopen(nome_file, "r");
                if (puntat == NULL)
                     printf ("SERVER: archivio non aperto\n");
                ret = stat(nome_file, &pro);
                fsize = (long)pro.st_size;
```

```
buf_file = malloc (sizeof(char)*(fsize+1));
fread(buf_file,1,fsize,puntat);
buf_file[fsize]='\0';

fclose(puntat);

chdir("..");


strncpy(nome_file, nome, sizeof(nome_file));
if(compressor == GNUZIP)
     strncat(nome_file, ".tar.gz",sizeof(nome_file)-strlen(nome_file)-1);
else
     strncat(nome_file, ".tgz", sizeof(nome_file)-strlen(nome_file)-1);


mes_len = strlen(nome_file);
ret = send(argos->sock_cl, &mes_len, sizeof(mes_len),0);
if(ret<sizeof(mes_len)){
     printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
     exit(1);
}

ret = send(argos->sock_cl, &nome_file, mes_len,0);
if(ret<mes_len){
     printf("ERRORE: impossibile inviare il messaggio");
     exit(1);
}
```

```
                mes_len = (unsigned int)fsize;
                ret = send(argos->sock_cl, &mes_len, sizeof(mes_len),0);
                if(ret<sizeof(mes_len)){
                     printf("ERRORE: impossibile inviare la lunghezza del messaggio");
                     exit(1);
                }

                ret = send(argos->sock_cl, buf_file, mes_len, 0);
                if(ret<mes_len){
                     printf("ERRORE: impossibile inviare il messaggio");
                     exit(1);
                }

                free(buf_file);

                ret = recv(argos->sock_cl, &mes_len, sizeof(mes_len), MSG_WAITALL);
                if(ret<mes_len){
                     printf("ERRORE: impossibile inviare il messaggio");
                     exit(1);
                }
                if (mes_len==0){
                     printf ("SERVER: problemi con la registrazione del file da parte del client.
Cancellazione files temporanei saltata\n");
                     strncpy(messaggio, "SERVER: errore lato client rilevato. File temporanei salvati.
Si prega di riprovare.\n", sizeof(messaggio)-1);
                     messaggio[1023] = '\0';


                     strncpy(nome_file, "./", sizeof(nome_file));
                     strncat(nome_file, tar_dir, sizeof(nome_file)-strlen(nome_file)-1);
```

```
        strncat(nome_file, "/", sizeof(nome_file)-strlen(nome_file)-1);
        strncat(nome_file, nome, sizeof(nome_file)-strlen(nome_file)-1);
        if(compressor == GNUZIP)
             strncat(nome_file, ".tar.gz",sizeof(nome_file)-strlen(nome_file)-1);
        else
             strncat(nome_file, ".tgz", sizeof(nome_file)-strlen(nome_file)-1);
        strncpy(messaggio, "./", sizeof(messaggio));
        //   strncat(messaggio, "/", sizeof(messaggio)-strlen(messaggio)-1);


        pid = fork();
        if(pid==0)
             execlp("/bin/rm", "rm", nome_file, NULL);
        wait(NULL);

        break;

}

printf("SERVER: file %s spedito con successo\n", nome_file);

pid = fork();
if (pid==0)
     execlp("/bin/rm", "rm", "-r", directory, NULL);
wait(NULL);

pid = fork();

if (pid==0)
     execlp("/bin/mkdir", "mkdir", directory, NULL);
```

```
                wait(NULL);


                pid = fork();
                if (pid==0)
                     execlp("/bin/rm", "rm", "-r", tar_dir, NULL);
                wait(NULL);

                pid = fork();

                if (pid==0)
                     execlp("/bin/mkdir", "mkdir", tar_dir, NULL);
                wait(NULL);

                vuoto = 1;

                printf("SERVER: file temporanei rimossi\n");

                strncpy(messaggio, "Archivio ", sizeof(messaggio)-1);
                messaggio[1023]='\0';
                strncat(messaggio, nome_file, sizeof(messaggio)-strlen(messaggio)-1);
                strncat(messaggio, " ricevuto con successo\n", sizeof(messaggio)-strlen(messaggio)-1);

                break;
           case 6:
                strncpy (messaggio, "Disconnessione in corso...\nGrazie per aver utilizzato Remote
Compressor\n", sizeof(messaggio)-1);
                messaggio[1023]='\0';

                pid = fork();
```

```
            if (pid == 0){
                 execlp("/bin/rm", "rm", "-r", directory,NULL);
            }
            wait(NULL);

            pid = fork();
            if (pid == 0){
                 execlp("/bin/rm", "rm", "-r", tar_dir,NULL);
            }
            wait(NULL);
            flag = 0;
            break;
 }

 /* Invio il messaggio al client */

 mes_len = strlen(messaggio);
 ret = send(argos->sock_cl, &mes_len, sizeof(mes_len),0);
 if(ret<sizeof(mes_len)){
      printf("ERRORE: impossibile inviare la lunghezza del messaggio");
      exit(1);
 }

 ret = send(argos->sock_cl, &messaggio, mes_len, 0);
 if(ret<mes_len){
      printf("ERRORE: impossibile inviare il messaggio");
      exit(1);
 }
}
```

```
    /* Chiudo il socket */

    close(argos->sock_cl);
    printf("SERVER: client %s disconnesso\n", cl_paddr);
    pthread_exit(NULL);
}

void* controllore (void* to_check){

    /* Thread che gestisce la terminazione dei thread
     che gestiscono le richieste */

    int t;
    int rc;
    void* status;
    t = *((int*)to_check);


    rc = pthread_join(t_id[t],&status);
    if (rc){
     printf("ERRORE: codice errato nel tentativo di join");
     exit(-1);
    }
    t_free[t] = 0;
    pthread_exit(NULL);
}

/* Funzione main() */
```

```c
int main (int argc, char* argv[]){
    struct sockaddr_in my_addr;
    int ret, sk, cn_sk, porta, optval, t;
    struct argomenti mess[BACKLOG_SIZE];
    int arg_cont[BACKLOG_SIZE];
    socklen_t len;

    /* Controllo il numero dei parametri */

    if (argc < 2){
     printf("COMPRESSOR-SERVER: Attenzione, sintassi errata.\n Eseguire il programma selezionando la
porta,\n esempio: >compressor-server 4000.\nImpossibile proseguire\n");
     return 0;
    }

    /* Converto a intero il nome della porta */

    porta = atoi(argv[1]);

    /* Creo il socket (PF_INET) TCP (SOCK_STREAM) */

    sk = socket(PF_INET, SOCK_STREAM, 0);
    if (sk == -1){
     printf("SERVER: Impossibile creare un nuovo socket sulla porta specificata.\n");
     return 0;
    }

    /* Setto l'attributo SO_REUSEADDR nel socket:
     SO_REUSEADDR: permette di fare una bind su una certa porta anche se
     esistono delle connessioni established che usano quella porta (il restart del
```

```
 server)
 SOL_SOCKET: e' un opzione di livello socket
 */

optval = 1;
ret = setsockopt(sk, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));
if (ret == -1){
 printf ("SERVER: Errore nel settaggio di SO_REUSEADDR.\n");
 return 0;
}

/*
 Con INADDR_ANY il server si mette in ascolto su una qualsiasi delle
 sue interfacce di rete
 */

memset(&my_addr, 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(porta);

/* Faccio la bind sul socket */

ret = bind(sk, (SA*) &my_addr, sizeof(my_addr));
if (ret == -1){
 printf ("SERVER: Impossibile eseguire il bind sul socket.\n");
 return 0;
}

printf("Aperta porta %d\n",porta);
```

```c
/* Faccio la listen sul socket */

ret = listen (sk, BACKLOG_SIZE);
if (ret == -1){
 printf("SERVER: Impossibile creare la coda di backlog lunga %d", BACKLOG_SIZE);
 return 0;
}

printf("REMOTE COMPRESSOR SERVER, V0.1\nIn attesa di connessioni...\n");

for(t=0;t<BACKLOG_SIZE;t++)
 t_free[t]=0;

pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);



while(1){

 /* Prelevo le connessioni dei client con la accept() */

 t=0;

 /* Mi procuro l'indice di un thread libero del pool di thread */

 while(t_free[t]!=0)
      t = (t + 1) %  BACKLOG_SIZE;
```

```
t_free[t] = 1;
arg_cont[t] = t;

/* accept():
 int accept(int sd, struct sockaddr* addr, socklen_t* addrlen);
 Restituisce il descrittore del connected socket
 */

len = (socklen_t) sizeof(mess[t].c_addr);
cn_sk = accept(sk, (SA*) &mess[t].c_addr, &len);

mess[t].sock_cl = cn_sk;

if (cn_sk == -1){
     printf ("SERVER: Errore in attesa di connessioni. Funzione accept()");
     return 0;
}


/* Faccio partire un thread che gestisce la richiesta a cui passo i parametri
 che gli servono (connected socket e indirizzo client */

if (pthread_create(&t_id[t],&attr,gestisci_richiesta,(void*) &mess[t])){
     printf("SERVER: Impossibile avviare un thread dal pool");
     return 0;
}

/* Faccio partire un thread controllore con gli argomenti del controllore */

if(pthread_create(&daeid[t],&attr,controllore,(void*) &arg_cont[t])){
```

```
            printf("SERVER: Impossibile creare thread di controllo");
            return 0;
        }


    }


}
```

```c
// compression-client v.0.1

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <arpa/inet.h>

# define SA struct sockaddr

int main (int argc, char* argv[]){
  int ret, sk, porta, t, s, try_send;
  unsigned int lun, mes_len;

  /* Creazione struttura per indirizzo server */

  struct sockaddr_in srv_addr;


  char comando [1025];
  char selezione [1025];
  char nome_file[1024];
  char *messaggio;
  char *buf_file;

  FILE *puntat;
```

```
  struct stat pro;
  int fsize;
  size_t f_check;

  const char *esecuzione[] = {"help","configure-compressor", "configure-name","show-
configuration","send","compress","quit","errore"};
  const char *compressore[] = {"bzip2", "gnuzip"};


  /* Verificare che il numero di parametri passati sia tre (nome del programma, IP, Porta) */

  if (argc != 3){
    printf ("Remote-Compressor: Parametri mancanti. Utilizzare la sintassi ->compressor-client
<indirizzo IP> <porta>\n\n");
    return 0;
  }

  /* Conversione a intero della porta */

  porta = atoi(argv[2]);

  /* Creazione del socket : int socket(int domain, int type, int protocol); */

  sk = socket(PF_INET, SOCK_STREAM, 0);
  if (sk == -1){
    printf("CLIENT: impossibile aprire il socket");
    return 0;
  }

  /* Azzero la memoria all' indirizzo srv_addr,
```

```
 * Inizializzo i campi della struttura e mi
 * preoccupo di convertire la porta in rappresentazione
 * network (Big Endian).
 * Con la inet_pton assegno al campo sin_addr della struttura
 * srv_addr l'indirizzo IP passato da main().
 */

memset(&srv_addr,0,sizeof(srv_addr));
srv_addr.sin_family = AF_INET;
srv_addr.sin_port = htons(porta);
ret = inet_pton(AF_INET, argv[1], &srv_addr.sin_addr);
if (ret <= 0){
  printf("Remote-compressor: Indirizzo non valido\n");
  return 0;
}


/* Mi connetto al socket tramite la connect():
 * int connect(int socket, const struct sockaddr *address, socklen_t address_len);
 */

ret = connect(sk, (struct sockaddr*) &srv_addr, sizeof(srv_addr));
if (ret == -1){
  printf("Remote-compressor: Impossibile connettersi al server.\n");
  return 0;
}

printf ("REMOTE-COMPRESSOR client v0.1\n");
printf ("Connesso al server %s sulla porta %d\n", argv[1], porta);
printf ("Inserire comandi di lunghezza massima 1024 caratteri\n");
```

```c
/* Ciclo infinito in cui gestisco gli inserimenti da riga di comando da parte del client */
while(1){
  printf ("Remote-compressor> ");
  scanf ("%s", comando);


   /* Cerco un comando valido tra quelli possibili, se non lo trovo dico che il comando
    * non e' valido
    */

  for (t = 0; t < 7; t++){
    if(!strcmp(comando, esecuzione[t]))
  break;
  }

  if (t == 7){
    printf("CLIENT: comando non valido. \nDigitare il comando help per avere aiuto.\n");
    continue;
  }

  /* Mando sul socket il comando richiesto dal client:
   * ssize_t send(int socket, const void *buffer, size_t length, int flags);
   */


  ret = send(sk, &t, sizeof(t),0);
  if (ret < sizeof(t)){
    printf ("CLIENT: Impossibile inviare il comando per intero");
```

```
  exit(1);
}


/* Qui comincia l'implementazione dei comandi */

switch (t){
 case 0: /* Comando help */
    break;
 case 1: /* Comando configure-compressor */
      scanf ("%s", selezione);
      for (s = 0;s < 2; s++){
           if (!strcmp(selezione,compressore[s]))
                break;
      }

      /* Invio il nome del compressore al server */

      ret = send(sk, &s, sizeof(s),0);
      if (ret < sizeof(s)){
           printf ("CLIENT: Impossibile inviare il comando per intero");
           exit(1);
      }

 break;
 case 2: /* Comando configure-name */

      /* Invio la lunghezza del nome  */

      scanf ("%s", selezione);
```

```
        mes_len = strlen(selezione);
        ret = send(sk, &mes_len, sizeof(mes_len),0);
        if(ret<sizeof(mes_len)){
             printf("ERRORE: impossibile inviare la lunghezza del messaggio");
             exit(1);
        }

        /* Invio il nome sul socket */

        ret = send(sk, &selezione, mes_len, 0);
        if(ret<mes_len){
             printf("ERRORE: impossibile inviare il messaggio");
             exit(1);
        }
        break;

case 3: /* Comando show-configuration */
        break;
case 4: /* Comando send */

        try_send = 1;
        while (try_send){
             scanf ("%s", selezione);
             if (!strcmp(selezione,"Q")){

                  /* Se è Q, abort del comando send */

                  try_send = 0;
                  mes_len = 0;
                  ret = send(sk,&mes_len,sizeof(mes_len),0);
```

```
                    if(ret<sizeof(mes_len)){
                         printf("ERRORE nella comunicazione con il server");
                         exit(1);
                    }
                    continue;
               }

               /* Apro il file passato dall' utente nella send e controllo se c'è
                * e se ho i permessi in lettura
                */

               puntat = fopen(selezione,"r");
               if (puntat == NULL){
                    printf("Attenzione!! File non trovato oppure permessi non validi. Reinserire il
nome del file, oppure inserisci Q\n");
                    /*invio codice di errore al server*/
                    continue;
               }

               /* Tramite la stat() mi procuro le informazioni sul file,
                * in particolare la lunghezza in byte
                */

               ret = stat(selezione,&pro);
               fsize = pro.st_size;

               /* Alloco *dinamicamente* un buffer delle dimensioni del file
                * e tramite la fread metto il file nel buffer
                */
```

```
                buf_file = malloc (sizeof(char)*(fsize + 1));
                if (buf_file==NULL){
                     printf("Errore nell'allocazione spazio dinamico in memoria\nReinserisci nome file
oppure digita Q> ");
                     continue;
                }
                f_check = fread(buf_file,1,fsize,puntat);
                buf_file[fsize]='\0';

                if (f_check!=fsize){
                     printf("Attenzione! Errore in lettura file.\nReinserire nome file oppure digita
Q> ");
                     continue;
                }

                fclose(puntat);

                /* Invio tramite socket della lunghezza del messaggio (nome del file) e del messaggio
*/

                mes_len = strlen(selezione);
                ret = send(sk, &mes_len, sizeof(mes_len),0);
                if(ret<sizeof(mes_len)){
                     printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
                     exit(1);
                }

                ret = send(sk, &selezione, mes_len,0);
                if(ret<mes_len){
                     printf("ERRORE: impossibile inviare il messaggio");
```

```
                exit(1);
            }


            /* Invio tramite socket della lunghezza del file e del file */

            mes_len = (unsigned int)fsize;
            ret = send(sk, &mes_len, sizeof(mes_len),0);
            if(ret<sizeof(mes_len)){
                printf("ERRORE: impossibile inviare la lunghezza del messaggio");
                exit(1);
            }

            ret = send(sk, buf_file, mes_len, 0);
            if(ret<mes_len){
                printf("ERRORE: impossibile inviare il messaggio");
                exit(1);
            }
            free(buf_file); /* LIBERO LA MEMORIA!!! */
            try_send = 0;
        }

        break;
    case 5:/* Comando compress */

        /* Prelevo il path e ricevo la dimensione del nome del file compresso dal server */

        scanf("%s", selezione);
        ret = recv(sk, &mes_len, sizeof(mes_len),MSG_WAITALL);
        if(ret<sizeof(mes_len)){
```

```
     printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
     exit(1);
}
if (mes_len==0)
     break;

/* Ricezione nome del file compresso dal server */

ret = recv(sk, &nome_file, mes_len, MSG_WAITALL);
if(ret<mes_len){
     printf("ERRORE: impossibile inviare il messaggio");
     exit(1);
}
nome_file[mes_len]='\0';

/* Ricevo dimensione del file dal server */

ret = recv(sk, &mes_len, sizeof(mes_len),MSG_WAITALL);
if(ret<sizeof(mes_len)){
     printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
     exit(1);
}

/* Alloco dinamicamente la memoria e ricevo il file compresso dal server */

buf_file = malloc(sizeof(char)*(mes_len + 1));
if (buf_file==NULL){
     printf("Attenzione! Errore nell'allocazione memoria dinamica");
     exit(2);
}
```

```
        ret = recv(sk, buf_file, mes_len, MSG_WAITALL);
        if(ret<mes_len){
             printf("ERRORE: impossibile inviare il messaggio\n");
             exit(1);
        }
        buf_file[mes_len]='\0';


        /* Aggiungo la "/" al path se non è presente */

        printf("Scrittura file %s in corso\n", nome_file);
        if (selezione[(sizeof(selezione)-1)] != '/')
             strncat(selezione,"/", sizeof(selezione)-strlen(selezione)-1);

        strncat(selezione, nome_file, sizeof(selezione)-strlen(selezione)-1);

        /* Apro il file in scrittura per scriverci il file compresso mandato dal server */

        puntat = fopen(selezione,"w");
        if (puntat==NULL){
             printf("Attenzione: problemi con il path selezionato, directory inesistente, oppure
non si possiedono i permessi per aprire il file.\n");
             mes_len = 0;
             ret = send(sk, &mes_len, sizeof(mes_len),0); /* Comunico l'errore al server */
             if(ret<sizeof(mes_len)){
                  printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
                  exit(1);
             }
             break;
```

```
        }

        /* Scrivo sul file */

        fsize = (int)mes_len;
        ret = fwrite (buf_file,1,fsize,puntat);
        if (ret!=fsize)
             printf("Errore nel salvataggio file\n");
        fclose(puntat);
        free(buf_file); /* LIBERO LA MEMORIA DINAMICA!!! */

        mes_len = 1;
        ret = send(sk, &mes_len, sizeof(mes_len),0);
        if(ret<sizeof(mes_len)){
             printf("ERRORE: impossibile ricevere la lunghezza del messaggio");
             exit(1);
        }


        break;
   case 6: /* Disconnessione dal server */
        break;
}


    /* Ricevo messaggio finale dal server */

    ret = recv(sk, &lun, sizeof(lun), MSG_WAITALL);
    if (ret != sizeof(lun)){
     printf ("CLIENT: Errore 1 nella comunicazione da parte del server");
```

```
        exit(1);
       }
       messaggio = malloc(sizeof(char) * (lun + 1));

       ret = recv (sk, messaggio, lun, MSG_WAITALL);
       if (ret != lun) {
        printf ("CLIENT: Errore 2 nella comunicazione da parte del server");
        exit(1);
       }
       messaggio[lun]= '\0';
       printf ("%s", messaggio);

       free(messaggio);
       if (t==6){ /* Se ho un quit, chiudo il socket e esco */
        close(sk);
        exit(0);
       }
  }


}
```