

```

/* Server Select (versione 1)*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define DIM_BUFF 100
#define LENGTH_FILE_NAME 90
#define LENGTH_PAROLA 30
#define max(a,b) ((a) > (b) ? (a) : (b))

/*****/

void gestore(int signo)
{
    int stato;
    printf("esecuzione gestore di SIGCHLD\n");
    wait(&stato);
}

/*****/

/* Legge una alla volta le linee del file restituendole in
 * come argomento di uscita, fino a EOF. Al raggiungimento
 * della fine del file viene restituito un puntatore a NULL
 */
char* leggiLinea(int fd)
{
    char* linea = malloc(DIM_BUFF+1);
    char c;
    int letti, i=0;

    while( (letti=read(fd, &c, 1) != 0) && (c!='\n') )
    { linea[i]=c; i++; }
    linea[i]='\0';

    if( letti == 0 ) return NULL;
    else return linea;
}

/*****/

/* Estrae dal datagramma ricevuto il nome del file e la parola.
 * Quindi controlla che almeno una riga del file contenga la
 * parola cercata e restituisce l'esito del controllo.
 */
int controlla_fileParola (char *input)
{
    char nome_file[LENGTH_FILE_NAME+1], parola[LENGTH_PAROLA+1];
    char* linea;
    int i=0, j=0, fd, ris=-1;
    while( input[i]!='\0' )
    { nome_file[j]=input[i]; i++; j++; }
    nome_file[j]=input[i];

```

```

i++; j=0;
while( input[i]!='\0' )
{ parola[j]=input[i]; i++; j++; }
parola[j]=input[i];

printf("Nome file: %s\n", nome_file);
printf("Parola: %s\n", parola);

fd=open(nome_file, O_RDONLY);
if (fd<0)
{ printf("File inesistente\n"); ris=-2; }
else
{
    while( (linea = leggiLinea(fd)) )
    {
        if( strstr(linea, parola) )
        { free(linea); ris=0; break; }
        else free(linea);
    }
    close(fd);
}

ris=htonl(ris);
return ris;
}

/*****

int main(int argc, char **argv)
{
    int listenfd, connfd, udpfd, fd_file, nready, maxfdpl;
    const int on = 1;
    char nome_file[LENGTH_FILE_NAME+1], parola[LENGTH_PAROLA+1],
          buffer[LENGTH_FILE_NAME+LENGTH_PAROLA+2];
    char* linea;
    fd_set rset;
    int len, nwrite, port, length, ris;

    struct sockaddr_in cliaddr, servaddr;

    /* CONTROLLO ARGOMENTI ----- */
    if(argc!=2)
    {
        printf("Error: %s port\n", argv[0]);
        exit(1);
    }
    else port = atoi(argv[1]); // Aggiungere controllo porta

    printf("Server avviato\n");

    /* CREAZIONE SOCKET TCP ----- */
    listenfd=socket(AF_INET, SOCK_STREAM, 0);
    if (listenfd <0)
    {perror("apertura socket TCP "); exit(1);}
    printf("Creata la socket TCP d'ascolto, fd=%d\n", listenfd);

    /* INIZIALIZZAZIONE INDIRIZZO SERVER ----- */
    memset ((char *)&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(port);

```

```

if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on))<0)
{perror("set opzioni socket TCP"); exit(2);}
printf("Set opzioni socket TCP ok\n");

if (bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr))<0)
{perror("bind socket TCP"); exit(3);}
printf("Bind socket TCP ok\n");

if (listen(listenfd, 5)<0)
{perror("listen"); exit(4);}
printf("Listen ok\n");

/* CREAZIONE SOCKET UDP ----- */
udpfd=socket(AF_INET, SOCK_DGRAM, 0);
if(udpfd <0)
{perror("apertura socket UDP"); exit(5);}
printf("Creata la socket UDP, fd=%d\n", udpfd);

/* INIZIALIZZAZIONE INDIRIZZO SERVER E BIND ----- */
memset ((char *)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(port);

if(setsockopt(udpfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on))<0)
{perror("set opzioni socket UDP"); exit(6);}
printf("Set opzioni socket UDP ok\n");

if(bind(udpfd, (struct sockaddr *) &servaddr, sizeof(servaddr))<0)
{perror("bind socket UDP"); exit(7);}
printf("Bind socket UDP ok\n");

/* AGGANCIO GESTORE PER EVITARE FIGLI ZOMBIE ----- */
signal(SIGCHLD, gestore);

/* PULIZIA E SETTAGGIO MASCHERA DEI FILE DESCRIPTOR ----- */
FD_ZERO(&rset);
maxfdpl=max(listenfd, udpfd)+1;

/* CICLO DI RICEZIONE EVENTI DALLA SELECT ----- */
for(;;)
{
    FD_SET(listenfd, &rset);
    FD_SET(udpfd, &rset);

    if ((nready=select(maxfdpl, &rset, NULL, NULL, NULL))<0)
    {
        if (errno==EINTR) continue;
        else {perror("select"); exit(8);}
    }

    /* GESTIONE RICHIESTE DI GET DI UN FILE ----- */
    if (FD_ISSET(listenfd, &rset))
    {
        printf("Ricevuta richiesta di get di un file\n");

        len = sizeof(struct sockaddr_in);
        if((connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &len))<0)
        {

```

```

    if (errno==EINTR) continue;
    else {perror("accept"); exit(9);}
}

if (fork()==0)
{ /* processo figlio che serve la richiesta di operazione */
    close(listenfd);
    printf("Dentro il figlio, pid=%i\n", getpid());

    int readRes;

    while((readRes=read(connfd, &length, sizeof(int)))>0)
    {
        length=ntohl(length);

        if ( (readRes=read(connfd, &nome_file, length))<0)
        { perror("read"); break; }
        else if( readRes==0) // abbiamo raggiunto la EOF
        { printf("Ricevuto EOF\n"); break; }

        printf("Richiesto file %s\n", nome_file);

        // Lettura parola, lunghezza poi stringa
        if( (readRes=read(connfd, &length, sizeof(int)))<0 )
        {
            perror("read");
            break;
        }
        else if( readRes==0) // abbiamo raggiunto la EOF
        { printf("Ricevuto EOF\n"); break; }
        length=ntohl(length);

        if ( (readRes=read(connfd, &parola, length))<0)
        { perror("read"); break; }
        else if( readRes==0) // abbiamo raggiunto la EOF
        { printf("Ricevuto EOF\n"); break; }

        printf("Richiesta parola %s\n", parola);

        fd_file=open(nome_file, O_RDONLY);
        if (fd_file<0)
        {
            printf("File inesistente\n");
            length=-2;
            length=htonl(length);
            write(connfd, &length, sizeof(int) );
        }
        else
        {
            /* lettura da file e invio delle linee con parola */
            while( (linea=leggiLinea( fd_file ))!=NULL )
            {
                if( strstr(linea, parola) )
                {
                    length=strlen(linea)+1;
                    length=htonl(length);
                    // invio lunghezza linea e linea
                    if ( (nwrite=write(connfd, &length, sizeof(int)))<0 )
                    {perror("write"); break;}

                    if ( (nwrite=write(connfd, linea, strlen(linea)+1))<0 )
                    {perror("write"); break;}
                }
                free(linea); // libero le risorse occupate
            }
            // il file e' terminato, lo segnalo al client e libero le

```

```

        // risorse
        length=-1;
        length=htonl(length);
        // invio lunghezza linea e linea
        if ( (nwrite=write(connfd, &length, sizeof(int)))<0 )
        {perror("write"); break;}
        close(fd_file);

        printf("Terminato invio file\n");
    }//else

}

//while
    // Lettura nome file, lunghezza poi stringa
    if( readRes<0)
    { perror("read"); }
    else if( readRes==0) // abbiamo raggiunto la EOF
    { printf("Ricevuto EOF\n"); }

    printf("Figlio %i: chiudo connessione e termino\n", getpid());
    close(connfd);
    exit(0);

}

//figlio

/* padre chiude la socket dell'operazione */
close(connfd);

} /* fine gestione richieste di file */

/* GESTIONE RICHIESTE DI CONTEGGIO ----- */

if (FD_ISSET(udpfd, &rset))
{

    printf("Server: ricevuta richiesta di verifica parola in file\n");

    len=sizeof(struct sockaddr_in);
    if (recvfrom(udpfd, &buffer, sizeof(buffer), 0, (struct sockaddr *)&cliaddr,
&len)<0)
    {perror("recvfrom"); continue;}

    ris = controlla_fileParola(buffer);
    printf("Risultato del controllo: %i\n", (int)ntohl(ris) );

    ris=htonl(ris);

    if (sendto(udpfd, &ris, sizeof(ris), 0, (struct sockaddr *)&cliaddr, len)<0)
    {perror("sendto"); continue;}

} /* fine gestione */

} /* ciclo for della select */

}

```