

fileInformation



[scopri il codice](#)

Studenti:

- Brachetta Matteo
- Maiellaro Giuseppe

fileInformation

“Dato un percorso in input restituisce le informazioni del file.”

Nel dettaglio restituirà le seguenti informazioni:

- Numero di caratteri
- Nome del proprietario
- Data ultima modifica
- Numero di righe
- Dimensione del file
- Permessi del file

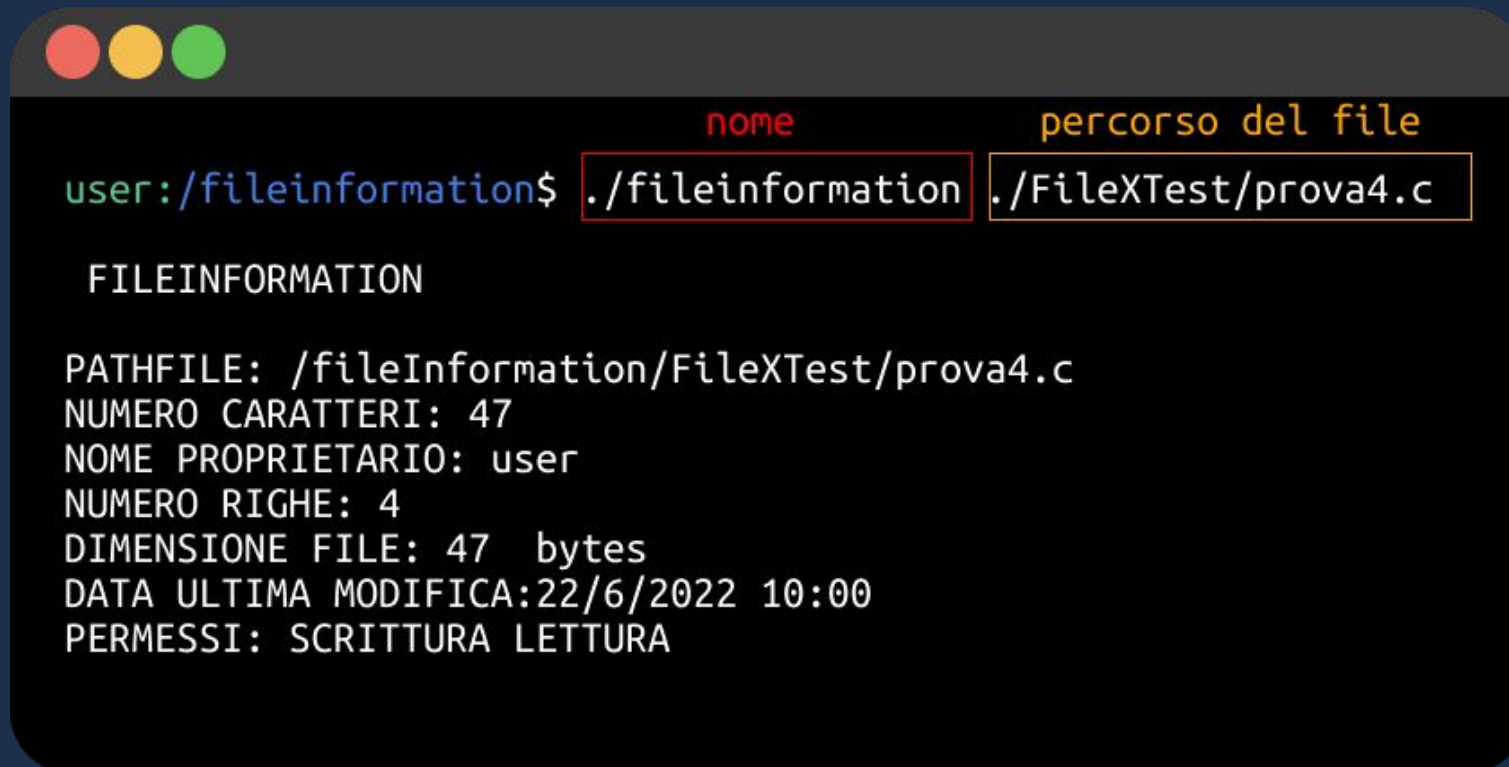
Configurazione



[vai al codice](#)

Default

Restituisce a video tutte informazioni del file.

A terminal window with a dark background and rounded corners. The title bar at the top has three colored buttons (red, yellow, green). The terminal shows a command prompt where the user has entered a command to get file information. The output displays various file attributes like path, character count, owner, line count, file size, modification date, and permissions.

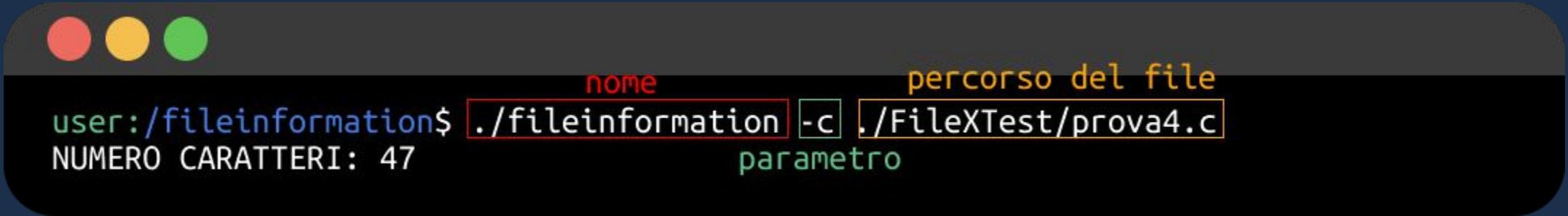
```
user:/fileinformation$ ./fileinformation ./FileXTest/prova4.c

FILEINFORMATION

PATHFILE: /fileInformation/FileXTest/prova4.c
NUMERO CARATTERI: 47
NOME PROPRIETARIO: user
NUMERO RIGHE: 4
DIMENSIONE FILE: 47 bytes
DATA ULTIMA MODIFICA:22/6/2022 10:00
PERMESSI: SCRITTURA LETTURA
```

Con parametri

Restituisce a video il risultato dalla richiesta del parametro



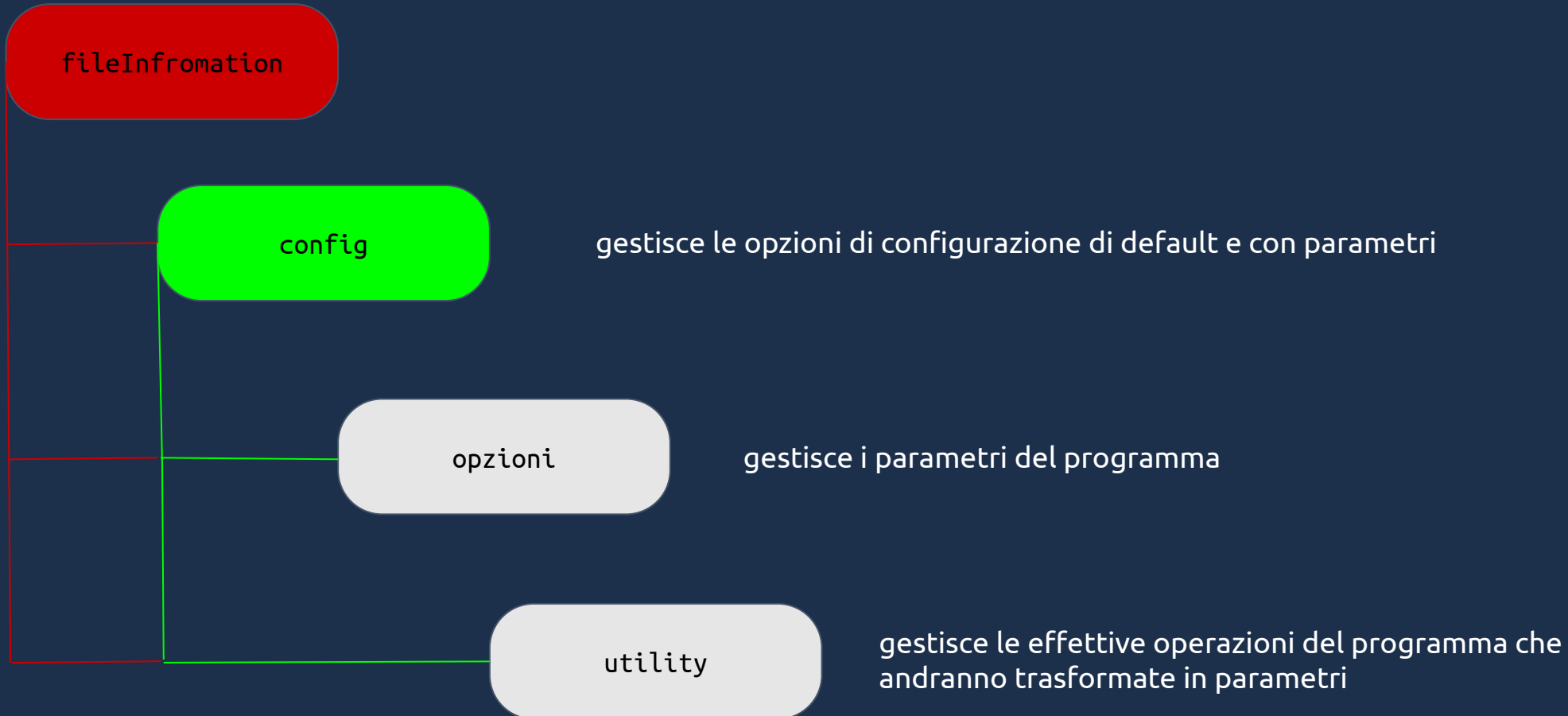
A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The prompt is `user:/fileinformation$`. The command entered is `./fileinformation -c ./FileXTest/prova4.c`. Annotations are present: `./fileinformation` is labeled "nome" in red; `-c` is labeled "parametro" in green; and `./FileXTest/prova4.c` is labeled "percorso del file" in orange. Below the command, the output is `NUMERO CARATTERI: 47`.

```
user:/fileinformation$ ./fileinformation -c ./FileXTest/prova4.c
NUMERO CARATTERI: 47
```

parametro	Descrizione	parametro	Descrizione
-c	restituisce il numero di caratteri	-s	restituisce la dimensione del file
-o	restituisce il nome del proprietario	-p	restituisce i permessi attribuiti al file
-d	restituisce la data di ultima modifica	-r	crea un report con tutti i dettagli del file, il file verrà chiamato report.txt
-l	restituisce il numero di righe	-a	restituisce tutte le informazione dei file presenti nella cartella in un file chiamato reportAnalisi.txt

Struttura

Struttura implementazione



Implementazione



[vai al codice](#)

Numero caratteri

```
int numeroCaratteri(char *pathFile)
{
    // file descriptor
    int fd;
    // buffer per read
    char buffer[MAX_BUFFER] = "\0";
    // contatore caratteri
    int caratteri = 0;
    if (verificaPercorso(pathFile))
    {
        // apertura file
        fd = open(pathFile, O_RDONLY);
        // lettura file
        read(fd, buffer, MAX_BUFFER);
        close(fd);
    }
    return strlen(buffer);
}
```

controllato il percorso del file, tramite le system call **open**, **read** e **close** rispettivamente otterremo l'apertura del file, la lettura ed il salvataggio dell'intero contenuto del file nella variabile **buffer** ed infine chiuderemo il file. Il numero totale di caratteri è ottenuto tramite il metodo **strlen** che restituirà la lunghezza del **buffer**.

Nome proprietario

```
char *nomeProprietario(char *pathFile)
{
    struct stat buf;
    struct passwd *pwd;
    // ottengo informazioni file
    stat(pathFile, &buf);
    // convero l'id in nome utente
    pwd = getpwuid(buf.st_uid);

    return pwd->pw_name;
}
```

tramite la system call `stat` si ottiene la struct `stat` che fornisce informazione sui file. In questo caso restituisce l'identificativo del file, che tramite la funzione `getpwuid` lo trasforma in nome utente

Dimensione file

```
int dimensioneFile(char *pathFile)
{
    int fd, size;
    if (verificaPercorso(pathFile))
    {
        fd = open(pathFile, O_RDONLY);
        // punto all'ultimo byte del file
        // così da ottenere la dimensione
        size = lseek(fd, 0, SEEK_END);
        close(fd);
    }
    return size;
}
```

per determinare la dimensione del file, utilizziamo la system call `lseek` che determina la posizione del puntatore. Tramite `SEEK_END` posizioniamo il puntatore alla fine del file e il risultato sarà la dimensione del file.

Data ultima modifica

```
time_t dataUltimaModifica(char *pathFile)
{
    struct stat buf;
    stat(pathFile, &buf);
    struct tm dataCreazione;
    // ottengo data ultima modifica del file
    dataCreazione = *(gmtime(&buf.st_mtim));
    // trasformo type tm in type time_t
    return mktime(&dataCreazione);
}
```

tramite la system call **stat**
otteniamo informazioni del file.
tramite le funzioni gmtime
e mktime trasformiamo il tipo in
time_t

Numero righe

```
int numeroRighe(char *pathFile)
{
    int fd, i, righe;
    char buffer[MAX_BUFFER];
    if (verificaPercorso(pathFile)){
        fd = open(pathFile, O_RDONLY);
        read(fd, buffer, MAX_BUFFER);
        close(fd);
        righe = 0;
        i = 0;
        // scorro fino alla fine del file
        while (buffer[i] != '\0')
        {
            // conto righe
            if (buffer[i] == '\n')
                righe++;
            i++;
        }
    }
    return righe;
}
```

tramite la system call `read` legge il file e salva il contenuto nella variabile `buffer`. Esaminando la variabile `buffer` e contando i caratteri speciali `\n` otterremo il numero delle righe

Permessi del file

```
char *permessi(char *pathFile)
{
    char *permessi;

    permessi = malloc(MAX_DIM_PERMESSI * sizeof(int));
    // controllo se la var contiene elementi in casi
    // di risposta affermativa elimino il contenuto
    if (strlen(permessi) > 0){
        free(permessi);
        permessi = malloc(MAX_DIM_PERMESSI * sizeof(int));
    }

    if (verificaPercorso(pathFile)){
        if (access(pathFile, 01) == 0)
            strcat(permessi, "ESECUZIONE ");

        if (access(pathFile, 02) == 0)
            strcat(permessi, "SCRITTURA ");

        if (access(pathFile, 04) == 0)
            strcat(permessi, "LETTURA ");
    }
    return permessi;
}
```

tramite la system call `access` verifico i permessi del file. Una volta verificati aggiungo alla variabile, liberata precedentemente, `permessi` il testo che rappresenta il permesso

Report file cartella

```
int visitaRicorsiva(char *filePath)
{
    struct dirent *dp;
    DIR *dir;
    char percorsoAggiuntivo[MAX_DIM_FILEPATH];
    // controllo che la cartella viene aperta correttamente
    if ((dir = opendir(filePath)) != NULL)
    {
        // leggo tutti gli elementi della cartella
        while ((dp = readdir(dir)) != NULL)
        {
            // controllo no siano cartelle "speciali"
            if (dp->d_name[0] != '.')
            {
                // azzero pathfile
                strcpy(percorsoAggiuntivo, "");
                // creo nuovo percorso
                // aggiungo la posizione attuale
                strcat(percorsoAggiuntivo, filePath);
                // aggiungo il nome della cartella del file da analizzare
                strcat(percorsoAggiuntivo, "/");
                strcat(percorsoAggiuntivo, dp->d_name);
                // verifico se è una cartella
                if (verificaCartella(percorsoAggiuntivo))
                {
                    // è una cartella richiamo la funzione sulla sottocartella
                    visitaRicorsiva(percorsoAggiuntivo);
                }
                else
                {
                    // è un file richiamo la configurazione di default per ottenere tutte l'informazioni
                    scriviNelFile(PERCORSOREPORTCARTELLA, configurazioneDefault(percorsoAggiuntivo));
                }
            }
        }
        closedir(dir);
    }
    return -1;
}
```

tramite la system call `readdir` scorro ogni elemento della cartella. Controllo che non siano cartelle speciali ed inizio la verifica. Se è un file scriverò nel report i dettagli altrimenti richiamerò la funzione.

Verifica percorso

```
int verificaPercorso(char *pathFile)
{
    int fd;
    if ((fd = open(pathFile, O_RDONLY)) != -1)
    {
        close(fd);
        return 1;
    }
    else
    {
        perror("ERRORE apertura file: ");
        return -1;
    }
}
```

tramite la system call **open** verifica che il file esiste altrimenti restituirà un errore

Verifica cartella

```
int verificaCartella(char *filePath){  
    DIR *dir;  
    if ((dir = opendir(filePath)) != NULL){  
        closedir(dir);  
        return 1;  
    }  
    else  
        return 0;  
}
```

tramite la system call `opendir` verifica l'esistenza della cartella. Se la cartella non esiste restituirà `0`

Crea report analisi

```
int creaReportAnalisi(){
    int fd;
    if ((fd = open(PERCORSOREPORTCARTELLA, O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU)) != -1)
    {
        close(fd);
        return 1;
    }
    perror("ERRORE creazione report");
    return -1;
}
```

Crea o ripristina il file
per immagazzinare il
report della cartella

Scrivi nel file

```
int scriviNelFile(char *pathFile, char *testo)
{
    int fd;
    if ((fd = open(pathFile, O_WRONLY | O_APPEND, S_IRWXU)) != -1)
    {
        if (write(fd, testo, strlen(testo)) != -1)
        {
            close(fd);
            return 1;
        }
        else
        {
            perror("ERRORE scrittura file:");
            return -1;
        }
    }
    else
    {
        perror("ERRORE apertura file:");
        return -1;
    }
}
```

una volta aperto il file in **APPEND** tramite la system call **write** inserisce il testo nel file.

Riepilogo

System call

System call	Tipo	Utilizzo
open, read e close	gestione file e directory	per gestire un file
opendir, readdir e closedir	gestione file e directory	per gestire le cartelle
access	gestione file e directory	per ottenere i permessi del file
lseek	gestione file e directory	per ottenere la dimensione del file
stat	gestione file e directory	informazioni del file

***Grazie per
l'attenzione***