Esempio

martedì 18 aprile 2023 15:50

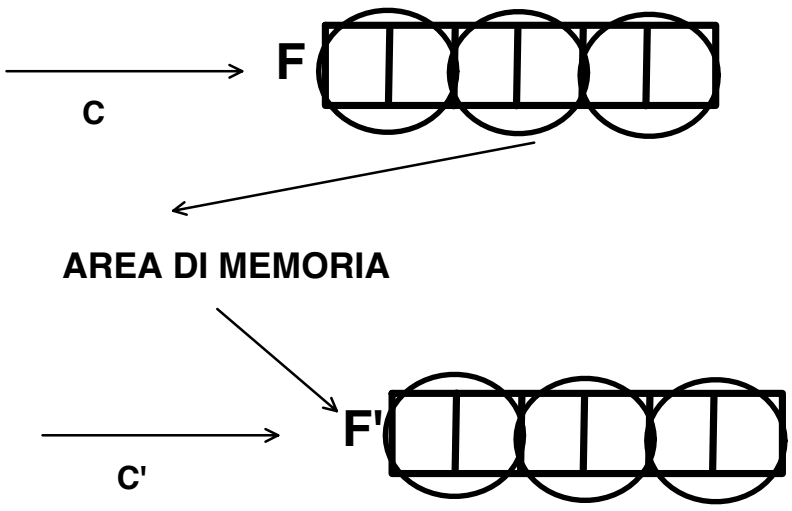
ESEMPIO

Abbiamo un file F e questo lo vogliamo copiare in un altro file. Il contenuto di F' è una copia del contenuto di F.

copy.c

```c
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

#define BUFSIZE 1024

int main(int argc, char *argv[]) {
    int sd, dd, size, result;
    char buffer[BUFSIZE];

    if (argc != 3) { /* check the number of arguments */

       printf("usage: copy source target\n");
       exit(1);
    }

    /* read only opening of the source file */
    sd=open(argv[1],O_RDONLY);
    if (sd == -1) {
       printf("source file open error\n");
           exit(1);
    }

    /* destination file creation */
    dd=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0660);
    if (dd == -1) {
       printf("destination file creation error\n");
           exit(1);
    }


   /* let's start with the copy operations */
   do {

           /* read up to BUFSIZE from source */
          size=read(sd,buffer,BUFSIZE);
          if (size == -1) {
            printf("source file read error\n");
            exit(1);
       }

       /* write up to BUFSIZE to destination file */
       result = write(dd,buffer,size);
       if (result == -1) {
                   printf("destination file write error\n");
              exit(1);
              }

   } while(size > 0);

       close(sd);
   close(dd);

}/* end main*/
```

Tutti i byte che sono presenti all'interno del file F, devono andare a finire all'interno di un nuovo file F'.
Devo prendere i byte F, portarli all'interno della mia area di memoria, e poi andarli a scrivere in F'.

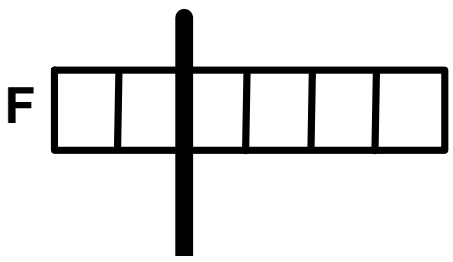


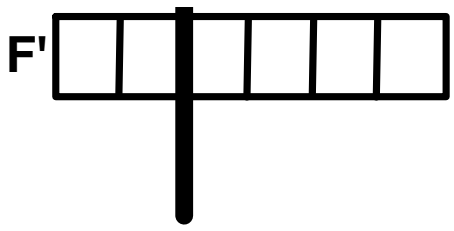
Per far questo devo avere un canale C per arrivare su F, e prendere i dati in lettura da F, e un canale C' per arrivare su F' per scrivere i dati all'interno di F'.
Se il file F è troppo grande non possiamo permetterci di caricare tutto insieme all'interno di un buffer e poi scaricarlo in F'. Leggiamo un blocco, lo carichiamo, e poi lo scarichiamo. E così via sino alla fine.

Nella riga 19 riceviamo in argv[1] il nome del file sorgente, nella riga 26 in argv[2] il nome del file destinazione, la sorgente la dobbiamo aprire con la funzione open e passiamo O_RDONLY, vogliamo solo leggere dalla sorgente.
Ci viene restituito un codice registrato all'interno della variabile sd, e controlliamo anche se la system call ha fallito. Così abbiamo il canale per arrivare a leggere.
Andiamo in open anche su argv[2] (riga 26), lo creiamo se non esiste, se esiste ne tronchiamo il contenuto perché dobbiamo eseguire la copia del contenuto di un altro file, e poi vogliamo solo scrivere in questo file.
Queste sono le informazioni che specifichiamo tramite la pipe | .
Se il file viene creato i permessi saranno 0660, RW a me, RW al gruppo, nulla agli altri.
Ovviamente se c'è un errore, usciamo.
Ora nel DO-WHILE utilizziamo un'area di memoria BUFFER di 1024 bytes, in cui un volta ci serve per leggere dal canale in cui abbiamo associato il file sorgente, scarichiamo i dati all'interno di quell'array quindi passiamo il puntatore dentro la read, e poi specifichiamo che vogliamo leggere BUFSIZE dati. La read ci restituisce in size, quanti bytes sono stati effettivamente consegnati, e a questo punto chiamiamo poi la write sulla destinazione dd, prendiamo i dati dallo stesso array buffer dove sono stati depositati, e poi specifichiamo che vogliamo scrivere stavolta solo size dati.
ATTENZIONE: Quando siamo arrivati alla fine del file, il puntatore punterà proprio alla fine di esso: a quel punto la sessione mi consegnerà size = 0, e li capisco che ho finito il ciclo. Perché ogni volta che leggo man mano il file pointer viene aggiornato.
Anche nella write ci ritornerà un valore che è il numero dei bytes effettivamente scritti ogni volta, quindi l'operazione di scrittura ha un residuo. Dipende dal driver che gestisce l'oggetto I/O.

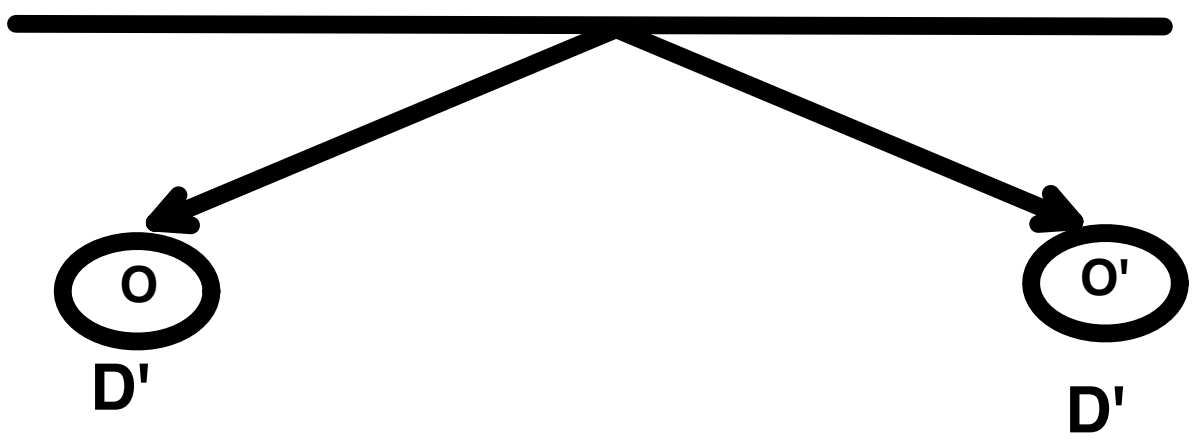
```
PLES/VIRTUAL-FILE-SYSTEM/UNIX/copy-command> ./a.out copy.c pippo
francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEA
PLES/VIRTUAL-FILE-SYSTEM/UNIX/copy-command> diff copy.c pippo
francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEA
```

Se siamo allineati nella sessione tipicamente il residuo non avviene, in questo esempio siamo perfettamente allineati tra file F e F':





Se noi chiediamo di scrivere SIZE bytes sul file di destinazione, quelli che avevamo caricato nel buffer di memoria volatile, e ci dice quanti di questi byte che io avevo chiesto che fossero scritti sono stati REALMENTE SCRITTI.
QUINDI NOI ABBIAMO UN'AREA in cui abbiamo inserito i bytes che vogliamo riportare sul file, e supponiamo che quest'area sia di taglia T e noi vogliamo andare a scrivere tutti T bytes all'interno del file.
Quando chiamo la write e cerco di portare questi dati sul file è possibile che, di questi bytes, soltanto un sottoinsieme sono realmente consegnati sul file. Chiaramente io posso accorgermi di questo, perché se ho chiesto di scrivere T bytes e mi accorgo dal valore di ritorno che sono stati scritti una quantità inferiore di byte, significa che l'operazione di scrittura che io ho chiamato ha un residuo. Bisogna gestire questi residui.
Il residuo è la parte che non è stata ancora scritta. Questo può capitare perché tutto dipende da qual è il driver d CHE USIAMO PER LAVORARE SULL'OGGETTO DI I/O CHE STIAMO GESTENDO.
Se noi lavoriamo su un oggetto di I/O O, poi lavoriamo su un oggetto di I/O O', quindi chiamiamo qualche system call che va a lavorare o su O o su O', magari una write (può essere la stessa system call), a seconda del canale andiamo da una parte o da un'altra parte, internamente il software del kernel significa che andrà a lavorare con un driver D associato all'oggetto O, oppure con un driver D' associato all'oggetto O'. Questi driver sono componenti software che hanno una logica interna differente - useranno aree di memoria in maniera differente - e quindi è possibile che se noi chiediamo la scrittura sull'oggetto O, questa scrittura vada a buon fine per tutti i byte che stiamo proponendo verso l'oggetto O.



Se o' è un socket, i dati vanno verso la rete, e il suo driver attualmente ha liberi un buffer di un sottoinsieme di dati che vogliamo scrivere. Il driver per ora accetta la scrittura di una quantità di byte inferiore, e cosa faccio? Ci riprovo. Scrivo anche quella che manca in un'altra write.
Devo essere in grado di strutturare il software in questo modo.
Il residuo lo gestiamo "ITERATIVAMENTE": Prima o poi questi dati verranno scritti.

12:32

DF è il comando che ci lista i file System su Linux.