

ESEMPIO



All'interno della cartella abbiamo vari programmi: uno è analyser.c, esso viene lanciato e prende come input il nome di un file, viene aperto il file e ci viene detto quante linee sono presenti sul file (una linea è tutta una sequenza di informazioni che abbiamo fino allo \n) e quante parole abbiamo.

Quindi ci va ad effettuare un'analisi sul file.

Writer.c fa questo:

Abbiamo lo standard input STDIN e lo standard output STDOUT: il programma non fa altro che leggere i byte da stdin e riemetterli in stdout, fa l'echo su standard output di tutti i byte che arrivano da stdin. Abbiamo però anche la gestione del residuo: se noi leggiamo 1024 da STDIN e cerchiamo di scrivere 1024 su stdout e non ci riusciamo, ciò che non abbiamo scritto riproviamo a scriverlo.

Redirector.c non fa altro che ridirezionare alcuni canali (in particolare un canale) e poi lanciare writer in una exec, il che implica dire che il processo è lo stesso. Apriamo un file, ridirezioniamo lo standard output su quel file e poi lanciamo writer, a questo punto il processo non fa altro che prendere dati dallo standard input e mandarli sullo standard output, ma di fatto stiamo andando sul file perché abbiamo ridirezionato lo standard output su file.

WRITER.C

```

1 #include <unistd.h>
2
3 #define MAXBUF 4096
4
5 int main () {
6     char buffer[MAXBUF];
7     int res_r,res_w,prev_w;
8
9     res_r = read(0, buffer, MAXBUF);
10
11    while(res_r)
12    {
13        prev_w = 0;
14        res_w = 0;
15        do {
16            prev_w =prev_w +res_w;
17            res_w = write(1, &buffer[prev_w],res_r-prev_w); //gestione residuo di scrittura
18        } while (res_w+prev_w <res_r); //finchè ho un residuo devo continuare ad eseguire il loop
19
20        res_r=read(0, buffer, MAXBUF);
21    }
22
23
24
25    return 0;
26
27 }
28

```

Nel ciclo while andiamo a controllare il valore di res_r, ossia il risultato della lettura.

Leggiamo da stdin e scarichiamo i dati in buffer, leggendo al più 4096 byte, riga 9. Ci viene detto con res_r quanto abbiamo letto. Ora entriamo nel while e scriviamo su stdout, e quando abbiamo finito di scrivere tutto considerando anche i residui, nella riga 20 andiamo a leggere nuovamente da standard input.

La scrittura sulla gestione dei residui è fatta utilizzando il do-while, di tutti i res_r byte che ho letto inizialmente su stdout non ho scritto nessun byte (prev_w = 0) e quindi non ho alcun risultato della scrittura (res_w = 0) e non ho alcun residuo di questa scrittura. In un do-while andiamo a chiamare una write e cerchiamo di scrivere quant'è il risultato della lettura (res_r) meno quello che ho scritto già prima (prev_w), ossia res_r - prev_w.

Quindi vado a scrivere solo il residuo, e lo devo prendere da quel buffer NEL PUNTO CHE VA AD INDICARE dove la scrittura precedente si è fermata. Quindi la precedente write ha scritto un solo byte rispetto a 1024 (res_w quindi diventa 1) e devo andare lì a prendere il primo byte e non lo zero, nel mio buffer. Chiaramente devo passare il & per indicare la zona dell'area di memoria.

Quindi scrivo su stdout. Poi nel post condizionale vado a vedere se c'è ancora un residuo: se non c'è più un residuo vuol dire che ho scritto tutto, a quel punto esco dal DO-WHILE e vado ad effettuare un'altra lettura.

Se noi lo eseguiamo, non fa altro che prendere qualsiasi cosa da tastiera e riemetterlo su terminale.

```

LE-SYSTEM/UNIX/analyser> ./writer
ksmg;lrdmdlkgngflknbgflkbngfln
ksmg;lrdmdlkgngflknbgflkbngfln

```

REDIRECTOR.C

```

redactor.c
1 #include <unistd.h>
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <fcntl.h>
5 #include <stdlib.h>
6 #include <stdio.h>
7
8 int main (int argc, char *argv[])
9 {
10    int fd;
11
12    if (argc!= 2) {
13        printf("Syntax: write_on_file <file_name>\n");
14        exit(-1);
15    }
16
17    fd=open(argv[1], O_CREAT| O_TRUNC|O_WRONLY,S_IRUSR|S_IWUSR);
18
19    if (fd == -1) {
20        perror("Open error: ");

```

Nell'esecuzione mettiamo nome dell'eseguibile e nome del file dove noi vogliamo andare a ridirezionare le nostre scritture da terminale, quindi ciò che arriva da standard input.

Apriamo quel file, se non esiste lo creiamo, se esiste lo tronchiamo e vogliamo solo scriverci sopra.

Se non ci sono stati errori mandiamo in printf() qual è il file descriptor che c'è tornato, CHIUDIAMO LO STANDARD OUTPUT, QUINDI LA PRINTF DEVE ESSERE CHIAMATA PRIMA DELLA CLOSE, ALTRIMENTI NON RIUSCIREMO A VEDERE LE SCRITTE A VIDEO, poi duplichiamo il file descriptor, e dato che il canale con codice 1 è stato chiuso, FD verrà duplicato lì, e poi facciamo la exec di quel writer, quindi partirà il programma di prima che leggerà da stdin e scriverà su stdout.

Ma a questo punto lo stdout va su file.

```

21     exit(2);
22 }
23
24 printf("fd=%d\n", fd);
25 close (1);
26 dup(fd);
27 execve("./writer", NULL, NULL);
28 perror("Exec error: ");
29 exit(3);
30 }
```

```

LE-SYSTEM/UNIX/analyser> make
gcc analyser.c -o analyser
gcc redirector.c -o write_on_file
gcc writer.c -o writer
```

```

LE-SYSTEM/UNIX/analyser> ./write_on_file pippo
fd=3
lkdslnl
dfkndfndkjn
dfknbd bfdkb
df bfdfm bfdkm
```

```

lkdslnl
dfkndfndkjn
dfknbd bfdkb
df bfdfm bfdkm
pippo lines 1-4/4 (END)
```

Quindi questi dati li ritroveremo sul file PIPPO

ANALYSER.C

```

2   #include <unistd.h>
3   #include <sys/types.h>
4   #include <sys/stat.h>
5   #include <fcntl.h>
6   #include <stdlib.h>
7   #include <string.h>
8
9   #define MAX_BUFFER 1000
10
11  int error_function() {
12      printf("Syntax: analyzer [-p] <nome_file>\n");
13      exit(-1);
14 }
15
16
17  int main (int argc, char *argv[]){
18
19
20      int fd, res_r, index;
21      char buffer[MAX_BUFFER];
22      char *filename;
23      char nextchar;
24
25
26      int continuous = 0;
27      int temp_charnum=0, temp_linenum=0, temp_wordnum=0;
28      int charnum=0,linenum=0, wordnum=0;
29      int newword = 0;
30
31      if ((argc<2) || (argc>3)) error_function();
32      if ((argc== 2) && (argv[1][0] == '-')) error_function();
33      if ((argc== 3) && (strcmp(argv[1], "-p")!=0)) error_function();
34      if ((argc== 3) && (strcmp(argv[1], "-p")==0)) continuous= 1;
35      if (argc== 2) filename=argv[1];
36      else filename = argv[2];
37
38
39      fd= open(filename, O_RDONLY| O_EXCL);
40      if (fd== -1)
41      {
42          perror("Open error: ");exit(-2);
43      }
44
45      do {
46          res_r = 1;
47          temp_linenum=0; temp_wordnum=0; temp_charnum=0;
48          newword=1;
49          while (res_r){
50              res_r = read (fd,buffer,MAX_BUFFER);
51              index=0;
52              while(index <res_r) {
53                  temp_charnum++;
54                  if ((buffer[index] != '\n') &&(buffer[index] != ' '))
55                      if (newword){newword=0; temp_wordnum++;}
56                  } else newword=1;
57                  if (buffer[index] == '\n') temp_linenum++;
58                  index++;
59              }
60
61          if ((linenum!= temp_linenum) || (wordnum!= temp_wordnum) ||(charnum!= temp_charnum)) {
62              linenum = temp_linenum; wordnum = temp_wordnum;charnum = temp_charnum;
63              printf("Number of characters: %d; number of words:%d; number of lines:%d\n",
64              charnum,wordnum,linenum);
65          }
66          if (continuous) {sleep(1); lseek(fd, 0,SEEK_SET);}
67      } while(continuous);
68      exit(0);
69  }
```

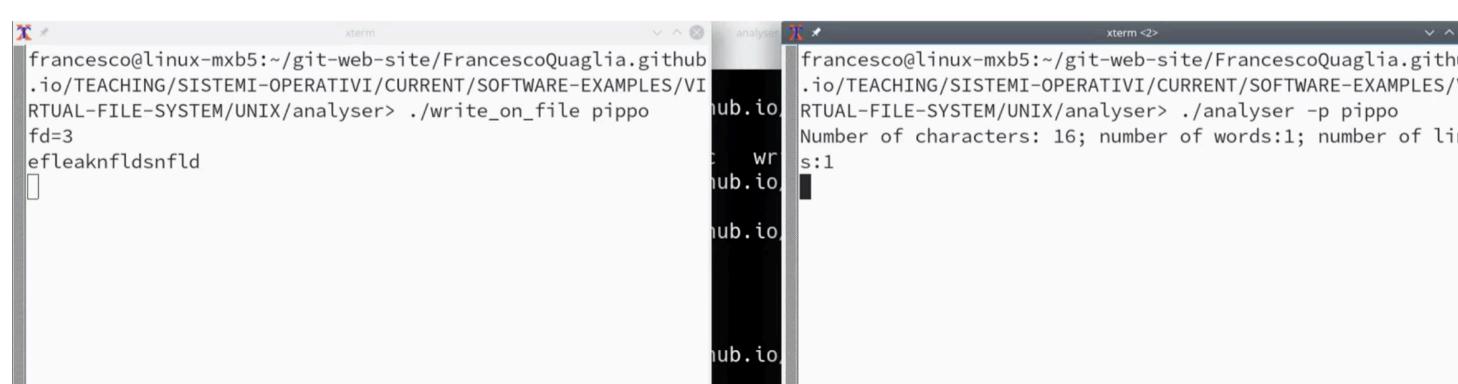
Qui controlliamo gli argomenti che ci vengono passati. In argv[1] ci aspettiamo il nome di un file che dobbiamo analizzare. Ma noi poi apriamo con una OPEN il file, ci prendiamo il canale per andare a lavorare su quel file, ma qui non chiediamo la creazione perché dobbiamo aprire un file per analizzarne il contenuto, quindi si presume che esista già.

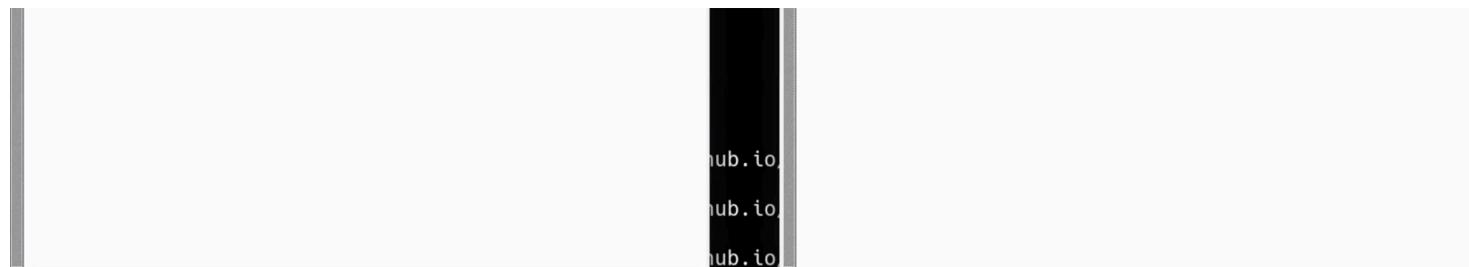
Ora in un DO-WHILE andiamo ad analizzare il contenuto e all'interno della zona del while c'è una lettura di dati che sono su quel file (FD) DI AL PIÙ dati MAX_BUFFER byte, e dopo che ci viene detto in res_r quanti sono questi dati che abbiamo letto, ci poniamo un indice a zero e analizziamo tutti i byte di quel buffer.

Finchè l'indice è minore di res_r andiamo a contare i caratteri e andiamo a vedere se il carattere in cui ci troviamo è uno \n o un blank. Qui di letture nel while ne effettuiamo più di una e andiamo avanti fino a che il risultato è diverso da zero della lettura.

Però siamo in un DO-WHILE e se la variabile continuous ha un valore di verità vero alla fine ritorniamo all'inizio del DO e ripartiamo.

Nel caso in cui noi siamo stati lanciati con 3 argomenti, e la prima stringa argv[1] era pari a "-p", ciò significa che l'analisi del file non va effettuata una volta sola, ma va effettuata periodicamente, e quindi se continuous è pari a 1, dopo aver controllato che il numero di byte che ho trovato, il numero di parole che ho trovato, il numero delle linee che ho trovato era diverso da quello che avevo trovato prima, dormo per un secondo e chiamo un lseek, per spostarmi su quel file che ho appena analizzato, a partire dall'inizio, e poi ricomincio ad analizzarlo.





Andiamo a cambiare il contenuto del file (ogni volta ci riposizioniamo all'inizio del file per controllare tutte queste situazioni).

The image shows two terminal windows side-by-side. The left window, titled 'xterm', displays the output of a C program named 'analyser'. It prints several strings from standard input: 'fd=3', 'efleaknfldsnfld', 'ldknfajfkfanf', 'a;kmfalkfnafn', and 'akfaf'. The right window, also titled 'xterm', shows the same program being run again. This time, it prints the analysis results for each line: 'Number of characters: 16; number of words:1; number of lines:1', 'Number of characters: 29; number of words:2; number of lines:2', 'Number of characters: 43; number of words:3; number of lines:3', and 'Number of characters: 49; number of words:4; number of lines:4'. Both terminals have a dark background with light-colored text.

```
francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEACHING/SISTEMI-OPERATIVI/CURRENT/SOFTWARE-EXAMPLES/VTUAL-FILE-SYSTEM/UNIX/analyser> ./write_on_file pippo
fd=3
efleaknfldsnfld
ldknfajfkfanf
a;kmfalkfnafn
akfaf
[

francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEACHING/SISTEMI-OPERATIVI/CURRENT/SOFTWARE-EXAMPLES/VTUAL-FILE-SYSTEM/UNIX/analyser> ./analyser -p pippo
Number of characters: 16; number of words:1; number of lines:1
s:1
Number of characters: 29; number of words:2; number of lines:2
s:2
Number of characters: 43; number of words:3; number of lines:3
s:3
Number of characters: 49; number of words:4; number of lines:4
s:4
[

|||ch
n = te
numbe
```

Considerazione: Noi stiamo lavorando sul file PIPPO. Ma in pratica quello che abbiamo detto a riguardo degli Hard-Links ci va ad indicare che il nome di un file non è una cosa così importante, perché lo stesso contenuto noi in realtà lo possiamo accedere utilizzando nomi diversi. È possibile avere uno scenario estremo, in cui un certo contenuto possiamo continuare ad utilizzarlo senza che ci sia alcun nome associato a lui?

```
LE-SYSTEM/UNIX/analyser> ls -lai
total 76
11797264 drwxr-xr-x  2 francesco users   4096 Apr 15 12:52 .
11797250 drwxr-xr-x 10 francesco users   4096 Apr 14 10:50 ..
11796822 -rwxr-xr-x  1 francesco users 12832 Apr 15 12:52 analyser
11797611 -rw-r--r--  1 francesco users 1682 Apr 15 11:12 analyser.c
11797608 -rw-r--r--  1 francesco users    92 Apr 15 11:19 Makefile
11797202 -rw-----  1 francesco users     49 Apr 15 12:59 pippo
11797660 -rw-r--r--  1 francesco users    510 Apr 14 10:55 redirector.c
11796823 -rwxr-xr-x  1 francesco users 12752 Apr 15 12:52 write_on_file
11796826 -rwxr-xr-x  1 francesco users 12440 Apr 15 12:52 writer
11797268 -rw-r--r--  1 francesco users    354 Apr 26 2018 writer.c
```

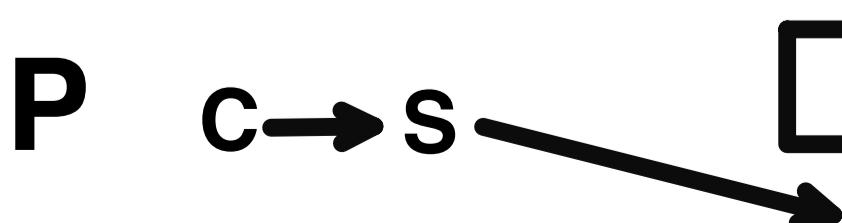
Questo file pippo ha un solo hard-link, rimuoviamo questo hard link con "rm pippo".
Quindi dobbiamo modificare la directory per dire che l'hard link non esiste più. Quindi il numero degli hard links che arrivano sull'I-NODE 11797202 è 0, però c'è un problema. Dopo tutto questo se continuiamo ad inserire dati queste applicazioni sono ancora attive.

```
francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEACHING/SISTEMI-OPERATIVI/CURRENT/SOFTWARE-EXAMPLES/VI/RTUAL-FILE-SYSTEM/UNIX/analyser> ./write_on_file pippo
fd=3
efleaknfldsnfld
ldknfajfkaf
a;kmfalkfnafn
akfaf
sgslnkgkfs
sld gsm gmsn g

francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEACHING/SISTEMI-OPERATIVI/CURRENT/SOFTWARE-EXAMPLES/VI/RTUAL-FILE-SYSTEM/UNIX/analyser> ./analyser -p pippo
Number of characters: 16; number of words:1; number of lines:1
Number of characters: 29; number of words:2; number of lines:2
Number of characters: 43; number of words:3; number of lines:3
Number of characters: 49; number of words:4; number of lines:4
Number of characters: 59; number of words:5; number of lines:5
Number of characters: 75; number of words:9; number of lines:6
```

Cosa sta succedendo?

Avevamo un file che si chiamava pippo ed era all'interno di una certa directory, quindi li c'era l'hard-link verso questo I-NODE. Avevamo aperto questo file pippo in due applicazioni. In un processo P abbiamo un canale C che ci porta su una sessione S che ci porta sull'i-node, in un processo P' abbiamo un canale C' con una sessione S' che ci porta sullo stesso i-node.





Supponiamo di rimuovere il collegamento di questo hard link appunto.

L'informazione è ancora utilizzabile, perché gli altri collegamenti la mantengono viva, ma non sono hard link, sono solo i riferimenti che queste sessioni hanno verso questo I-NODE.

Quindi l'I-NODE non viene rimosso. Quindi questo file continua ad esistere senza più un nome, perché all'interno dell'I-NODE il nome non c'è scritto, questo implica dire che io questo file non lo posso più riaprire usando una open, perché la open vuole sapere il nome del file ossia un hard link verso questo I-NODE, MA POSSO USARLO UTILIZZANDO QUESTE SESSIONI!

Quindi quando rimuoviamo i file, la rimozione del reale contenuto associato a quel file avverrà soltanto se non ci sono sessioni correntemente attive verso quei file.

In particolare quando rimuoviamo gli hard-link verso il file dobbiamo stare attenti anche a questi scenari.