I File in Unix

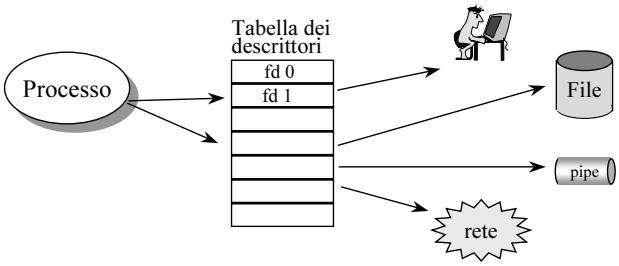
Un processo Unix vede tutto il mondo esterno (I/O) come un insieme di descrittori (da qui discende omogeneità tra file e dispositivi di Unix).

I file descriptor sono piccoli interi non negativi che identificano i file aperti

standard input, standard output, standard error sono associati ai file descriptor 0, 1, 2

Nuove operazioni di RICHIESTA producono nuovi file descriptor per un processo.

Numero massimo di fd per processo e per sistema



I processi interagiscono con l'I/O secondo il paradigma *open-read-write-close* (operazioni di prologo e di epilogo)

Flessibilità (possibilità di pipe e ridirezione)

System Call per operare a basso livello sui file
(`creat, open, close, read/write, lseek`)

Prologo (apertura/creazione di file):

CREATE fd = **creat**(name,mode);
int fd; /* file descriptor */
int mode; /* attributi del file */
⇨ diritti di UNIX (di solito espressi in ottale)
⇨ file name aperto in scrittura

OPEN fd = **open**(name, flag);
char *name;
int flag; /* 0 lettura, 1 scrittura, 2 entrambe */
int fd; /* file descriptor */

⇨ apre il file di nome **name** con modalità **flag**
⇨ in **/usr/include/fcntl.h** sono definite le costanti O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_CREAT, O_TRUNC, O_EXCL

Esempi

```
fd=open("file", O_WRONLY| O_APPEND)
fd=open("file", O_WRONLY| O_CREAT| O_APPEND, 0644)
fd=open("file", O_WRONLY| O_CREAT| O_TRUNC, 0644)
fd=open("lock", O_WRONLY| O_CREAT| O_EXCL, 0644)
```

Epilogo (chiusura di file):

CLOSE retval = **close**(fd);
int fd, retval;

Operazioni di RICHIESTA e RILASCIO risorse
(max num. fd aperti per processo e per macchina)

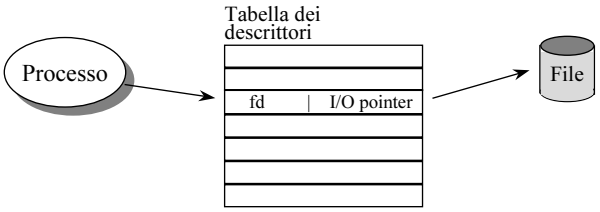
I FILE in UNIX

I file in Unix sono una sequenza di **BYTE**

ACCESSO **sequenziale**

I file sono rappresentati dai file descriptor

Presenza di **I/O pointer** associato al file (e al processo).
I/O pointer punta alla posizione corrente del file su cui il processo sta operando (scrivendo/leggendo)



File Descriptor

In generale, la lettura da fd 0 ⇒ legge da **standard input**
la scrittura su fd 1 ⇒ scrive su **standard output**
la scrittura su fd 2 ⇒ scrive su **standard error**

Questi tre **file descriptor** sono aperti *automaticamente* dal **sistema** (shell) per ogni processo e collegati all'I/O

Per progettare **FILTRI**
cioè usare RIDIREZIONE e PIPING

i filtri leggono direttamente dal file descriptor 0
scrivono direttamente sul file descriptor 1

Completa omogeneità dei file con i dispositivi

fd = open ("/dev/printer", O_WRONLY);

Anche per i dispositivi usiamo le stesse primitive
open, read, write, close

Operazioni di Lettura e Scrittura

READ nread = **read**(fd, buf, n);

WRITE nwrite = **write**(fd, buf, n);

```
int nread, nwrite, n, fd;
char *buf;
```

- **lettura e scrittura** di un file avvengono a partire dalla **posizione corrente** del file ed avanzano il puntatore (**I/O pointer**) all'interno del file
- restituiscono:
  il **numero dei byte** su cui hanno lavorato
  -1 in caso di errore (come tutte system call)

Ogni utente ha la **propria visione** dei file aperti:

⇨ Nel caso di più utenti che aprono lo stesso file, ogni processo utente ha un proprio I/O pointer separato

⇨ **SE** un utente legge o scrive, modifica solo il proprio pointer, non modifica l'I/O pointer di altri

FILE SYSTEM

Un utente non ha visibilità delle azioni di un altro utente

Esempi di lettura/scrittura

COPIA da un FILE a un ALTRO

```
#include <fcntl.h>
#include <stdio.h>
#define perm 0644

main  ()
{ char f1 [20]= "file",
       f2 [40]= "/temp/file2";
   int infile, outfile; /* file descriptor  */
   int  nread;
   char buffer [BUFSIZ];

infile = open   (f1, O_RDONLY);
outfile = creat   (f2, perm);

while((nread = read(infile, buffer, BUFSIZ)) > 0)
           write (outfile, buffer, nread );

close (infile);
close (outfile);
}
```

Legge dal file *file* e scrive su *file2* in /temp

Copia da un File a un altro (uso argomenti)

```
#define perm 0777
main  (argc, argv)
int argc;
char **argv;

{     int infile, outfile, nread;
   char buffer [15];

infile  = open (argv[1], 0);
outfile = creat (argv[2], perm);
while (( nread = read (infile , buffer, 1)) > 0 )
          write (outfile, buffer, 1 );

close (infile);
close (outfile);
}
```

Con RIDIREZIONE

```
#define LUNG 1
main  ()
{ char  buffer [LUNG];
    while ( read   (0, buffer, LUNG) > 0 )
              write  (1 , buffer, LUNG);
}
```

Il sistema esegue i collegamenti tra file descriptor e file

Copia file con controllo degli errori

```
#include <fcntl.h>
#include <stdio.h>
#define perm 0744  /* tutti i diritti all'owner
                lettura al gruppo ed altri   */

main  (argc, argv)
int argc;
char **argv;
{ int status;
  int infile, outfile, nread;
  char  buffer[BUFSIZ];  /*buffer  per  i  caratteri
  */

  if (argc != 3)
        { printf (" errore \n"); exit (1); }

  if ((infile=open(argv[1], O_RDONLY)) <0)
        exit(1); /* Caso di errore */

  if ((outfile=creat(argv[2], perm )) <0)
           {close (infile); exit(1); }

  while((nread=read(infile, buffer, BUFSIZ)) >0 )
  { if(write(outfile, buffer, nread)< nread)
    {close(infile);close(outfile);exit(1);}
                   /* Caso di errore */
  }
  close(infile); close(outfile); exit(0);
}
```

Efficienza delle system call **read** e **write:** dipendenza dalle dimesioni del buffer