Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

# Sistemi operativi e reti

A.A. 2016-17

Pietro Frasca

## Lezione 24

Giovedì 19-01-2017

- L’accesso al file avviene mediante le funzioni **read** (lettura) e **write** (scrittura):

```
int read (int fd, char *buffer, int n)
int write (int fd, char *buffer, int n)
```

- Le funzioni ritornano rispettivamente il numero di byte letti e scritti.
- Per l’accesso diretto (random) ai file si può usare la funzione lseek.

```
int lseek (int fd, int offset, int da_dove );
```

La funzione **lseek** riposiziona il puntatore del file avente descrittore *fd* alla posizione *offset* secondo il valore del parametro *da_dove,* il quale può assumere i seguenti valori:

**SEEK_SET**, il file pointer è impostato sulla posizione di *offset byte a partire dall'inizio del file;*

**SEEK_CUR**, il puntatore del file è impostato a offset byte a partire dalla sua posizione corrente;

**SEEK_END,** Il puntatore è impostato alla posizione *offset* byte a partire dalla sua dimensione (fine del file).

## Esempi di SC dei file

- Il primo esempio mostra la realizzazione di un comando **cp** (copia di file).
- Nel secondo esempio, si illustra il caso in cui due processi, padre e figlio, che accedono allo stesso file, ne condividono anche l'I/O pointer.
- Nel terzo esempio si mostra come realizzare l'accesso diretto mediante la lseek.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#define DIMBUF 1024
#define PERMESSI 0755

int main (int argc, char **argv){
  int stato,fin, fout,n;
  char buffer[DIMBUF];
  if (argc != 3){
    printf("errore \n");
    exit(1);
  }
  if ((fin=open(argv[1],O_RDONLY))<0){
     printf ("errore lettura file");
     exit(1);
  }
  if ((fout=open(argv[2],O_CREAT|O_WRONLY,PERMESSI))<0){
     printf ("errore scrittura file");
     exit(1);
  }
```

```
while ((n=read(fin,buffer,DIMBUF))>0)
    if (write(fout,buffer,n)<n){
      close(fin);
      close (fout);
      exit(1);
    }
  close(fin);
  close(fout);
  exit(0);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#define N 10
int main(){
  int i,fd1,fd2,pid;
  fd1=open("pub.txt",O_CREAT|O_WRONLY,0777);
  pid=fork();
  if (pid==0) {
    fd2=open("priv.txt",O_CREAT|O_WRONLY,0777);
    for (i=0;i<N;i++) {
      write (fd1,"figlio",6);
      usleep(100);
      write (fd2,"figlio",6);
    }
    close (fd2);
  }
  else if (pid>0) {
    for (i=0;i<N;i++) {
      write (fd1,"padre",5);
      usleep(100);
    }
    close(fd1);  }}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
struct Persona {
   int id;
   char cognome[40];
   char nome[20];
   char tel[16];
} persona;
int main(){
   int fd,size,i=0,n;
   size=sizeof(persona);
   fd=open("./persone.db",O_CREAT|O_RDWR);
   persona.id=10;
   strcpy(persona.cognome,"Perin");
   strcpy(persona.nome,"Stefania");
   strcpy(persona.tel,"06102030");
   lseek(fd,size*i,SEEK_SET);
   write (fd,&persona,size);
   i++;
```

```
    persona.id=11;
    strcpy(persona.cognome,"Rossini");
    strcpy(persona.nome,"Mario");
    strcpy(persona.tel,"338112233");
    lseek(fd,size*i,SEEK_SET);
    write (fd,&persona,size);
    i++;
    persona.id=12;
    strcpy(persona.cognome,"White");
    strcpy(persona.nome,"Roger");
    strcpy(persona.tel,"021112233");
    lseek(fd,size*i,SEEK_SET);
    write (fd,&persona,size);
    close(fd);
    fd=open("./persone.db",O_RDONLY);
    i=1; // posiziona la lettura al primo record
    lseek(fd,size*i,SEEK_SET);
    read(fd,&persona,size);
    printf("cognome %s\n",persona.cognome);
  close(fd);
}
```

# Protezione

- La protezione in unix/linux avviene a livello di:
  - **autenticazione degli utenti**
  - **controllo dell'accesso alle risorse**

- L'accesso degli utenti al sistema consiste nella verifica di due parametri: **username** e **password**. Ogni utente è identificato nel sistema mediante un **nome utente** a cui corrisponde uno **user-id** (dato di tipo intero) e una **password** (dato di tipo string).
- L'utente **root** (detto super-user) è l'utente che ha i privilegi di amministratore di sistema, il suo **user-id** ha valore **0**.
- Gli utenti sono aggregati in gruppi. Ogni gruppo è identificato da un **nome logico** a cui è associato un numero intero: il **group-id**. Ogni utente deve appartenere almeno ad un gruppo, detto **gruppo di default**, e può appartenere a più gruppi.

# Il file /etc/passwd

- L'elenco degli utenti è contenuto nel file di sistema **/etc/passwd**.
- Il file /etc/passwd ha un record composto da 7 campi; il carattere : (due punti) è il separatore di campo:
  1. User-name dell'utente
  2. Password crittografata
  3. User-id dell'utente
  4. Group-id a cui appartiene l'utente
  5. Descrizione dell'utente
  6. Home directory dell'utente
  7. Programma che viene eseguito al login

**Esempio:**

```
root:x:0:0:root:/root:/bin/bash
rossi:x:510:501:rossi lino:/home/rossi:/bin/sh
bianchi:x:511:501:bianchi eva:/home/bianchi:/bin/bash
```

# Il file /etc/group

- definisce i gruppi ai quali appartengono gli utenti. Ogni gruppo è specificato da un record, avente il carattere : (due punti) come separatore di campo, che ha il seguente formato:

  **nome_gruppo:passwd:GID:lista_utenti**

- **nome_gruppo** indica il nome logico del gruppo;
- **password** se specificata (non è obbligatoria) indica la password (criptata) del gruppo;
- **GID (Group IDentifier)** è l'identificativo numerico del gruppo.
- **lista_utenti** elenco dei nomi logici degli utenti, appartenenti ad un diverso gruppo di default, separati da virgole.

**Esempio di record di /etc/group:**

```
studenti::501
tesisti::502,bianchi, rossi
```

- L'accesso al sistema avviene tramite **il login**. L'utente deve digitare al prompt username e password che vengono confrontate con i rispettivi valori presenti nel file **/etc/passwd e /etc/shadow**.
- In unix l'accesso alle risorse è basato sulle liste di controllo degli accessi **ACL** (**A**ccess **C**ontrol **L**ist): per ogni risorsa sono definiti i diritti di accesso associati agli utenti. In unix le risorse sono viste come file.

- In unix gli utenti, per quanto riguarda la sicurezza, sono distinti in tre gruppi:
  1. **Utente** (**User**) (costituito dal solo proprietario del file)
  2. **Gruppo** (**Group**) (costituito da tutti gli utenti appartenenti al gruppo del proprietario)
  3. **Altri** (**Other**) (costituito da tutti gli altri utenti del sistema che non appartengono agli altri due gruppi precedentemente descritti)
- Il **proprietario** del file stabilisce i diritti di accesso per i vari gruppi, mediante il comando **chmod**. Ad esempio il comando `chmod 755 test.c`.

- Per ogni file sono ammesse tre modalità di accesso:
  - **Lettura:** il contenuto del file può essere solo letto
  - **Scrittura:** il file può essere cancellato e il suo contenuto può essere modificato.
  - **Esecuzione**: il file può essere eseguito: in questo caso il file deve essere o un programma eseguibile o uno script. Nel caso di directory la **x** ha il significato di permettere la visualizzazione del listato della directory.
- L'ACL di ogni file è composta da 9 bit, che indica i diritti di accesso per gli utenti del sistema, visti come appartenenti ai tre gruppi: **U**ser, **G**roup e **O**ther (UGO).

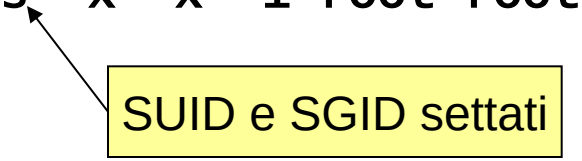
| r | w | x | r | w | x | r | w | x |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| **U**ser | | | **G**roup | | | **O**ther | | |

- Per ognuno dei tre gruppi i diritti si esprimono con tre bit nel ordine lettura, scrittura, esecuzione (read, write, execute) i cui simboli sono rispettivamente r, w e x.
- Oltre ai 9 bit che stabiliscono i permessi di accesso, per ogni file sono specificati altri 3 bit, che hanno significato solo nel caso in cui i file sono eseguibili (programmi o script):
  - Il bit SUID (**S**et **U**ser **ID**)
  - Il bit SGID (**S**et **G**roup **ID**)
  - Il bit STIcky (**S**ave **T**ext **I**mage)
- Infatti, quando si avvia un programma il sistema inserisce nel descrittore del nuovo processo che si crea lo **user-id** e il **group-id**, appartenenti all'utente che lancia il programma.
- E' possibile cambiare il proprietario del file settando i bit **SUID** e **SGID**. Infatti se il bit SUID è posto ad 1, al processo che esegue il file sarà assegnato lo **user-id** del proprietario del file (analogamente per SGID). In tal modo l'utente che lancia il file eseguibile assume, temporaneamente, l'identità del proprietario del file.

- Un esempio d'uso di SUID e SGID si ha nel programma **passwd**.
- Il comando passwd consente a un utente di cambiare la propria password, modificando il proprio record all'interno del file /etc/passwd, che per sicurezza appartiene all'utente root (superuser) e può essere modificato solo da questo. Il file eseguibile /bin/passwd ha entrambi i bit SUID e SGID settati ad 1 consentendo così ad un qualsiasi utente di assumere l'identità di root per la durata dell'esecuzione del programma e quindi di modificare il file /etc/passwd.
- Se si esegue il list del file si può vedere che è presente il carattere **s** minuscolo al posto della **x**.

- **[frasca@localhost]$ ls -l /usr/bin/passwd**

**-r-s--x--x  1 root root 16336 13 feb  2003 /usr/bin/passwd**



```
[frasca@localhost]$ ls -l /etc/passwd
-rw-r--r--  1 root root 21907 Dec  1 13:36 /etc/passwd
```

- Sui vecchi sistemi unix, se un file ha lo **sticky bit** settato ad 1, consente di mantenere memorizzato il codice del processo nell'area di swap, anche dopo la sua terminazione. Questo consentiva al programma di avviarsi più rapidamente. Questa caratteristica non è più usata nei moderni sistemi. Linux, ad esempio, ignora lo sticky bit sui file. Altri sistemi unix possono usare lo sticky bit sui file per scopi particolari. In alcuni sistemi, solo il superuser può settare lo sticky bit sui file. Quando lo sticky bit è usato su una directory, i file in quella directory possono essere eliminati o rinominati solo dal proprietario o da root. Senza lo sticky bit anche gli utenti che hanno accesso in scrittura a quei file possono cancellarli e rinominarli.

# Gestione degli errori

- La maggior parte delle system call restituisce il valore -1 in caso di errore ed assegna lo specifico codice di errore alla variabile globale

```
int errno;
```

- Se la system call ha successo**, errno** non viene resettato (mantiene l'ultimo codice di errore che si è verificato).

# Il file header errno.h

Il **file errno.h** contiene la definizione dei nomi simbolici dei codici di errore

**Esempio:**

```
# define EPERM 1 /* Not owner */
# define ENOENT 2 /* No such file or directory */
# define ESRCH 3 /* No such process */
# define EINTR 4 /* Interrupted system call */
# define EIO 5 /* I/O error */
...
```

# La SC perror

void **perror** (const char *prefisso)

- converte il codice in **errno** nel corrispondente messaggio di errore, e lo visualizza anteponendo la stringa prefisso.

**Esempio:**

...

- ```
  fd=open(“prova.txt", O_RDONLY);
  if (fd==-1) perror (“errore file");
  ...

  errore file: No such file or directory
  ```
  ```

# Shell in UNIX

- Una shell è un'interfaccia utente a riga di comando, tramite la quale è possibile effettuare la gestione e il controllo del sistema operativo. **Sh** è la shell originale di Unix, ed è la shell predefinita per molti sistemi Unix. Attualmente la shell di base di Linux è **bash**, una versione potenziata della sh.
- Tramite la shell è possibile avviare  i vari comandi del sistema operativo. I comandi sono un insieme di *utility* contenuti generalmente nella directory **/bin**. Attraverso i comandi è possibile gestire e controllare le varie componenti del sistema operativo studiate durante il corso come la gestione dei processi, il file system, la memoria, i dispositivi etc.
- La shell ha anche un insieme di istruzioni interne che consentono di realizzare utilissimi script di shell. Pertanto la shell può essere usata in modo interattivo o in modalità batch, lanciando uno script.

## Connessione al sistema

- Attualmente il modo più diffuso che un utente usa per connettersi ad un server Unix o Linux collegato in rete, consiste nel utilizzare un programma di emulazione terminale per la connessione remota.
- Originariamente l'utente usava l'applicazione telnet, che implementava l'omonimo protocollo di comunicazione. Attualmente telnet, poiché invia sulla rete dati in chiaro (non criptati), non garantisce la dovuta sicurezza necessaria per una connessione remota e pertanto è stato sostituito da applicazioni basate su **ssh** (security shell).
- Tramite ssh, un'utente fornisce i dati per la connessione che consistono nello specificare il nome o l'indirizzo IP del server Unix e le proprie credenziali, cioè username e password. Se il login ha esito positivo, l'utente può utilizzare la shell come interfaccia con in sistema operativo remoto.

# Gestione del file system

Ogni utente può gestire file e directory appartenenti al file system, compatibilmente con le politiche di protezione applicate ai file, utilizzando comandi specifici.

***Spostamento nella directory corrente.***

A ogni utente è associata una directory corrente che ne specifica la posizione corrente all'interno della struttura logica del file system. Il comando **cd** consente la *navigazione* del file system, modificando la directory corrente.

Esempi:

**cd miadir**
**cd so/processi**
**cd /home/bianchi**

Col primo comando la directory corrente diviene **miadir**. Ogni successivo riferimento a file o directory, espresso come nome relativo sarà implicitamente riferito alla directory *miadir*. L'argomento (miadir) può essere specificato sia in **formato relativo** che **assoluto**

## *Creazione di una nuova directory*

Il comando da utilizzare è **mkdir**:

**mkdir nuovadir**.

Questo comando crea la nuova directory *nuovadir*. Il comando può fallire se non si posseggono i diritti di scrittura nella directory di appartenenza della nuova directory.

## *Cancellazione di una directory*

Questa operazione si effettua mediante il comando **rmdir o rm**:

**rmdir vecchiadir**

**rm –r vecchiadir**

Il comando rmdir cancellerà la directory vecchiadir, a condizione che essa sia vuota.

## *Visualizzazione del contenuto di una directory.*

Il comando **ls** permette l'esplorazione di una directory, secondo la sintassi:

**ls miadir**

**ls –l miadir**

**ls –la miadir**

**ls –R miadir**

In questo modo si ottiene la visualizzazione sullo standard output dei nomi dei file e directory contenuti nella directory data.

Per questo comando sono molto utili l'opzione **-l** che fornisce la visualizzazione di vari attributi del file; l'opzione **–a** che permette anche la visualizzazione di file il cui nome inizia con il punto (dot file), usati generalmente come file di configurazione e l'estensione **-R** per la scansione ricorsiva della directory e di tutte le directory in essa contenute.

## *Visualizzazione del contenuto di un file*

Il comando **cat** consente la visualizzazione sullo standard output del contenuto di uno o più file secondo la sintassi:

**cat miofile**

**cat file1 file2 .. fileN**

Il comando visualizzerà il contenuto dei file specificati, secondo la sequenza indicata.

Un altro comando molto diffuso per la visualizzazione dei file è **more**.

Questo comando visualizza il contenuto del file una schermata alla volta.

**more miofile**

## *Copia di file e directory*

È possibile duplicare un file mediante il comando **cp**, seguendo la sintassi:

**cp sorgente destinazione**
**cp –r dir1 dir2**

In questo modo si ottiene la copia del contenuto del file dato come primo argomento (*sorgente*) in un file di nome *destinazione*. Se il file *destinazione* esiste, il suo contenuto sarà sovrascritto; altrimenti, il file destinazione sarà creato.

- Il comando *cp* si può usare anche per le directory, eventualmente usando l'opzione -r (copia ricorsiva di sottodirectory).

- Per esempio:

  **cp -r dir1 dir2**

  eseguirà la copia dell'intero contenuto di dir1 (compresi eventuali sottodirectory) nella directory dir2.

## *Link di file*

Una delle caratteristiche di Unix nella gestione dei file riguarda la possibilità di creare collegamenti (link), che consente di associare più nomi allo stesso file. L'aggiunta di un nuovo link è possibile tramite il comando **ln**:

**ln file_esistente file_nuovo**.
**ln –s miadir ~/info/2016/sor/eser/proc**

In questo modo si aggiunge il link *file_nuovo* al file di nome *file_esistente*. Da ora in poi, nomi file_esistente e file_nuovo rappresenteranno due alternative equivalenti per riferirsi allo stesso file.

## *Cancellazione di file*

Quando è necessario cancellare uno o più file dal file system si può utilizzare il comando **rm**. Per esempio

**rm miofile**

elimina il file **miofile** dalla directory alla quale appartiene. Se si cancella un link, rm cancellerà soltanto il nome del link specificato.

## *Protezione*

Per modificare il valore dei bit di protezione associati a un file si deve utilizzare il comando **chmod**:

**chmod diritti miofile**

**chmod 0755**
**chmod 755**

dove l'argomento diritti esprime i diritti che si vogliono attribuire al file miofile. Sono previsti diversi modi di esprimere i diritti: uno di questi consiste nella specifica mediante codifica ottale del valore dei 12 bit di protezione. Per esempio: chmod 0744 miofile rende il file leggibile, scrivibile ed eseguibile per l'utente proprietario e soltanto leggibile ed eseguibile per gruppo e altri.

# Monitoraggio e gestione dei processi

- In un sistema multiutente e multiprogrammato come Unix e Linux, ogni utente può avere la necessità di monitorare e controllare l'esecuzione dei processi.
- È' possibile ottenere informazioni relativamente ai processi attualmente presenti nel sistema mediante il comando **ps** (proces status). Questo comando, utilizzato eventualmente con alcune opzioni, mostra per ogni processo presente nel sistema alcune informazioni, come il PID e il PPID, lo stato e il nome del programma che il processo sta eseguendo. In particolare, mentre il default prevede la visualizzazione dei soli processi dell'utente che esegue il comando, l'opzione **-a** mostra informazioni su tutti i processi.

***Terminazione e sospensione di processi***

È possibile inviare segnali ai processi tramite il comando **kill**, che consente di interrompere o eventualmente terminare forzatamente l'esecuzione di un processo. La sintassi è la seguente:

**kill [-segnale] pid**
**kill -l**

L'argomento *pid* è l'identificatore del processo destinatario, mentre il parametro opzionale *segnale* è il segnale da inviare. In assenza dell'opzione, sarà inviato il segnale SIGTERM, che provoca la terminazione forzata del processo destinatario.

Inoltre, se si desidera sospendere la shell per un intervallo di tempo, il comando da usare è sleep, al quale si fornisce un argomento che rappresenta la durata dell'intervallo di tempo in secondi; per esempio sleep 60 sospende il processo per un minuto.

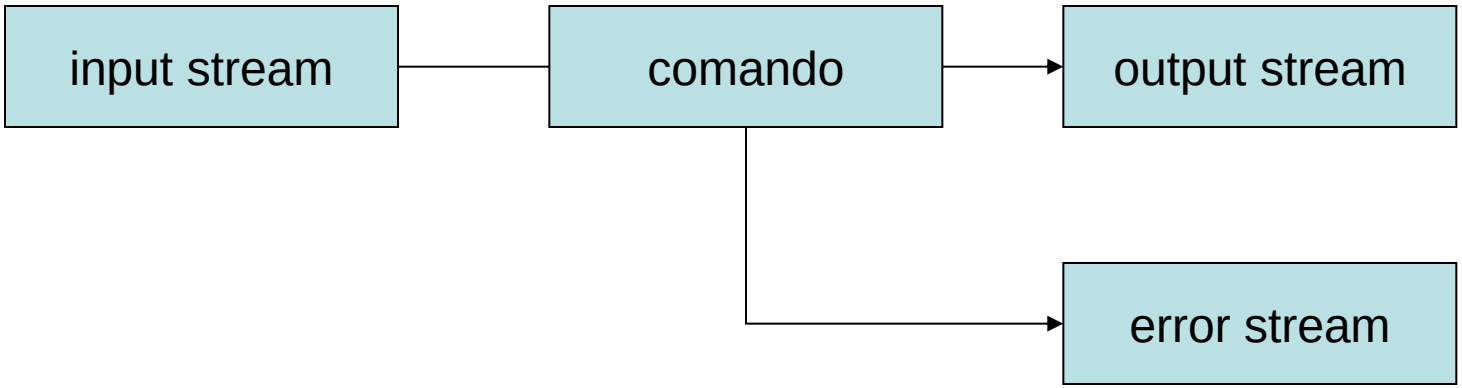
- Un utile comando è ***echo***, che invia sullo standard output la stringa data come argomento. Per esempio, il comando:

  **echo** "**ciao a tutti**"

  visualizza sullo schermo la stringa "ciao a tutti".

# Input/output dei comandi: ridirezione e pipe

- In generale, ogni comando può acquisire dati da un canale d’ingresso e inviare i suoi risultati in un canale d'uscita, come mostrato dalla figura seguente.



- Il canale di ingresso *(input stream),* è associato al dispositivo di standard input, che generalmente è la tastiera.
- Il canale di uscita *(output stream)* corrisponde al dispositivo di standard output, di solito, lo schermo.
- E' inoltre presente un secondo canale di uscita, detto *error stream,* che è associato al dispositivo di standard error, sul quale sono trasmesse eventuali notifiche di errore. Anche il dispositivo predefinito per lo standard error è generalmente lo schermo.
- Sebbene gli stream di input e di output per default siano associati rispettivamente ai dispositivi standard di input e output, è possibile modificare questa impostazione tramite gli operatori di ***ridirezione****.*
- In particolare, con l'operatore di **ridirezione in input <** è possibile assegnare un diverso canale rispetto allo standard. In questo caso la sintassi è la seguente:

  **comando < filein**

Se **filein** è un file, *comando* elaborerà il contenuto del file, anziché lo stream proveniente dallo standard input. Per esempio, supponendo di aver memorizzato nel file rubrica nomi e numeri di telefono il comando:

**grep lino < rubrica**

visualizzerà tutte le righe che contengono la parola lino.

In modo analogo con l'operatore di **ridirezione in output >** è possibile assegnare l'output su un canale diverso dallo standard output. In questo caso la sintassi è la seguente:

**comando > fileout**
**ls –l > elenco.txt**

nel primo esempio l'output del comando sarà scritto nel file *fileout* riscrivendone l'eventuale contenuto. Il secondo esempio memorizza nel file *elenco.txt* il contenuto della directory corrente.

- In alternativa, volendo conservare il contenuto di fileout, si può usare l'operatore **>>** :

  **comando >> fileout**

  aggiungerà l'output del comando in coda al contenuto di fileout.
- Gli operatori di ridirezione in input e output possono essere usati insieme, come mostrato nel esempio seguente:

  **grep pippo < rubrica > pippo.txt**

  che selezionerà dal file rubrica le linee di testo contenenti la parola pippo, salvando il risultato nel file pippo.txt.
- Quando l'output di un comando *com1* si usa come input da un altro comando *com2* si può procedere in due modi. Il primo modo usa un file temporaneo, ottenuto con gli operatori di ridirezione, con una sequenza di due comandi:

  **com1 > temp**
  **com2 < temp**

In questo caso i due comandi sono eseguiti sequenzialmente e il file *temp* è il mezzo di comunicazione tra di essi. In alternativa, per ottenere un maggior grado di concorrenza, Unix supporta l'operatore pipe, rappresentato dal simbolo **|**, che consente il concatenamento di due comandi secondo la sintassi **com1 | com2,** come ad esempio:

**man ps | grep user**

- L'effetto prodotto da questo comando sarà la ridirezione dell'output di *com1* nell'input di *com2* e, a differenza della modalità basata sul file temporaneo, i due comandi sono eseguiti da processi concorrenti. La comunicazione tra i due processi è realizzata da una ***pipe*** che viene creata dalla shell per questo scopo. Per esempio il comando:

**ls –R /home/pietro/sor | grep filosofi.c**

viene eseguito da 2 processi concorrenti che eseguono rispettivamente il comando ls e il comando grep. Il primo processo produce in uscita i nomi di file e directory contenuti nel sotto albero del file system che parte dalla directory */home/pietro/sor,* mentre il secondo processo elabora l'output del primo, selezionando solo le linee che contengono la stringa *filosofi*.c