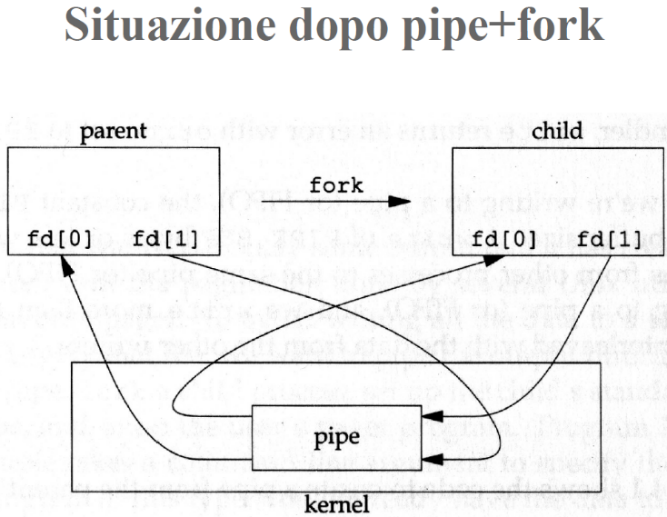
***Le pipe,*** laComunicazione tra Processi può avvenire:

* Passando dei files aperti tramite ***fork***;
* Attraverso il ***filesystem***;
* Utilizzando le ***pipe***;
* Utilizzando le ***FIFO***;
* Utilizzando ***IPC di System V***;
* Utilizzando ***stream*** e ***socket***.
* int ***pipe***(int *filedes*[2] );
* *filedes*[0] è il file descriptor di un "file" aperto in lettura;
* *filedes*[1] è il file descriptor di un "file" aperto in scrittura;
* inoltre l'output di *filedes*[1] corrisponde all'input di *filedes*[0].

*Restituisce* 0 se OK, -1 in caso di errore.

**UTILIZZO DELLE PIPE:**

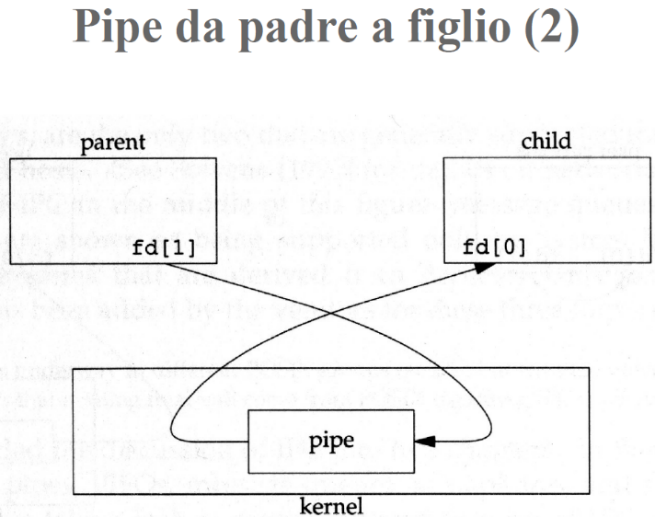
1.

int fd[2];

…

pipe(fd);

pid=fork();

…

2.

if(pid>0) { // padre

close(fd[0]);}

else if(pid==0) { // figlio

close(fd[1]);}

* Questo crea un canale

dal padre verso il figlio

3.

Una volta che è stata creata la ***pipe*** e che è stato scelto il verso di comunicazione è possibile utilizzare le funzioni di I/O che lavorano con i file descriptor (tranne *open*, *creat* e *lseek*).

Una ***pipe*** è un canale di comunicazione in cui i dati vengono letti nello stesso ordine in cui vengono scritti.

**I/O SU UNA PIPE:**

* Funzione ***write:***

Quando la pipe si riempie (la costante PIPE\_BUF specifica la dimensione), la ***write*** si blocca fino a che la ***read*** non ha rimosso un numero sufficiente di dati.

Se il descrittore del file che legge dalla pipe è chiuso, una ***write*** genererà un errore (segnale SIGPIPE).

* Funzione ***read:***

Legge i dati dalla pipe nell’ordine in cui sono scritti. Non è possibile rileggere o rimandare indietro i dato letti.

Se la pipe è vuota la ***read*** si blocca fino a che non vi siano dei dati disponibili.

Se il descrittore del file in scrittura è chiuso, la ***read*** restituirà un errore dopo aver completato la lettura dei dati.

* Funzione ***close:***

La funzione ***close*** sul descrittore del file in scrittura agisce come *end-of-file* per la ***read***.

La chiusura del descrittore del file in lettura causa un errore nella ***write***.

...

pid=fork();

if(pid>0) { /\* padre \*/

close(fd[0]);

write(fd[1], "ciao figlio\n",12);

}else{ /\* figlio \*/

close(fd[1]);

n=read(fd[0], line, 12);

write(STDOUT\_FILENO, line, n);

}

**UTILIZZO DELLE PIPE:**

Una cosa interessante è duplicare i descrittori della pipe su standard input ed output, a questo punto il figlio esegue con una exec un programma che può leggere da standard I/O, ad esempio more è uno di questi programmi.

# cat file | more

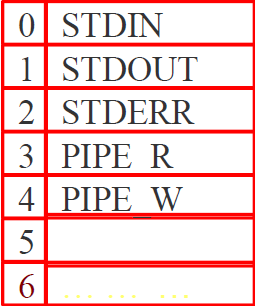
Il comando cat ha come input *il file* e come output *STDOUT*.

Il comando more ha come input *un file* e come output *STDOUT*.

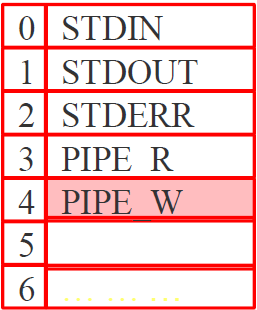
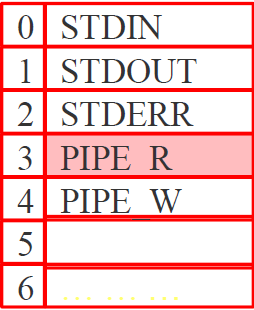
**ESEMPIO:**

**1.FIGLIO 2.FIGLIO**

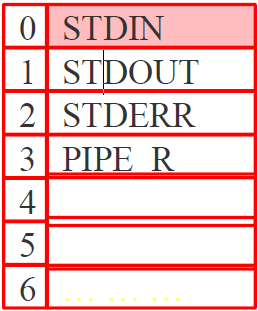
Nel processo *shell* viene creata la ***pipe***  Uguale a 1.FIGLIO

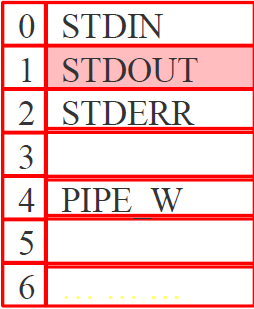
***pipe(fd);***

Dopo l’esecuzione delle ***fork***, Viene chiuso il descrittore del file in

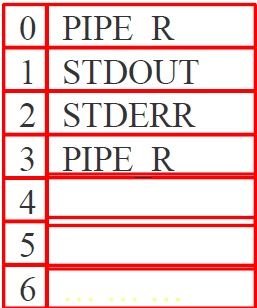
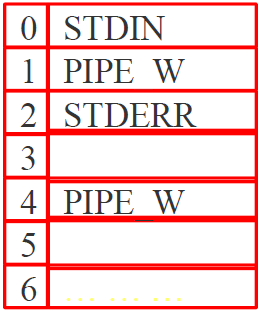
********nel primo figlio viene chiuso il *descrittore in lettura* scrittura

***close(fd[0]); close(fd[1]);***

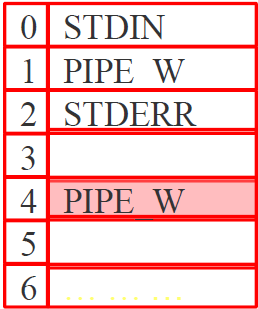
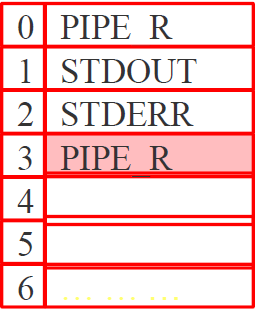
******Successivamente viene chiuso lo *standard output*  Viene chiuso lo *standard input*

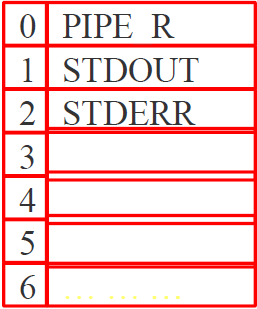
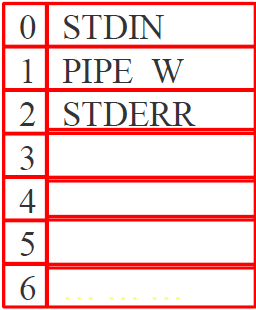
*****close(1); close(0);***

Con la funzione ***dup*** lo *standard output* coinciderà Con la ***dup*** lo *standard input* coincide

******con il descrittore del *file in scrittura* con il descrittore del *file in lettura*

***dup(fd[1]); dup(fd[0]);***

**Si chiude il descrittore del *file in scrittura* Si chiude il descrittoredel *file in lettura*

******Si esegue la ***exec*** del comando ***cat*** Si esegue la ***exec*** del comando ***cat***

***exec(cat); exec(more);***