Comunicazione tra processi: pipe

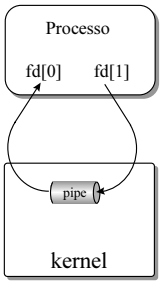
Le **pipe** sono un meccanismo UNIX di
Inter Process Communication (**IPC**)

- Le pipe sono canali di comunicazione **unidirezionali**
- **Limitazione** pipe: permettono la comunicazione solo tra processi che hanno un qualche avo in comune

Creazione di una pipe

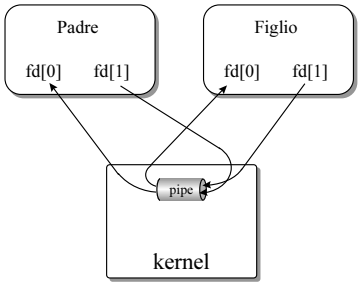
PIPE
```
#include<unistd.h>
retval = pipe (fd);
int retval;
int fd[2];
```

retval ritorna 0 in caso di successo, altrimenti un valore negativo.
in caso di successo,
vengono creati due file descriptor fd[0] e fd[1], rispettivamente, per lettura e per scrittura sulla pipe

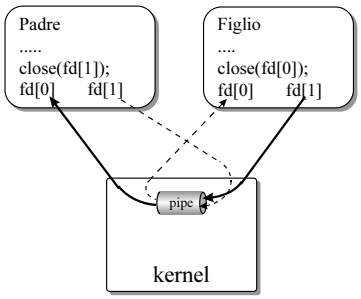


Comunicazione via pipe

Normalmente, un processo **apre una pipe**, quindi **genera un figlio** (che avrà i fd del padre duplicati). Padre e figlio possono comunicare via pipe.



Problema: padre e figlio potrebbero entrambi scrivere o leggere sulla pipe ➔ ogni processo chiude un lato della pipe e usa solo l'altro



Lettura/Scrittura su pipe

Come sui file, si usano **READ** e **WRITE** per leggere o scrivere dei dati sulla pipe

Differenze nell'uso di **read** e **write**
nelle **pipe** rispetto ai **file**

Nella **pipe** è insito un **meccanismo di sincronizzazione** tra lettore e scrittore, per cui:

➔ **read** blocca il processo lettore se la pipe è vuota (cioè sospende il processo lettore in attesa di dati)

➔ **write** blocca il processo scrittore se non c'è spazio dentro la pipe (cioè sospende il processo scrittore fino a che il lettore non libera spazio sufficiente dalla pipe)

Nelle pipe non c'è un I/O pointer, scrittura e lettura FIFO.

Chiusura pipe

CLOSE per chiudere un estremo della pipe

La chiusura di uno degli estremi di una pipe **non** provoca altre azioni fino a che ci sono **altri** processi che hanno aperto lo **stesso** estremo della pipe

Se un processo chiude l'**ultimo** estremo (di scrittura o di lettura) di una pipe:

- **lettura da pipe chiusa**
  ➔ **read** ritorna **0**, ad indicare la **fine del file**

- **scrittura su pipe chiusa**
  ➔ **write** provoca l'invio di **SIGPIPE** al processo scrittore (write ritorna -1
  errno=EPIPE "broken pipe").

Cosa succede ai processi eventualmente sospesi su una read da una pipe vuota o sospesi su una write su una pipe piena se viene chiuso l'altro estremo della pipe?

Differenze pipe - file

a) a un file descriptor non corrisponde alcun nome nel file system
   ➔ la pipe è una struttura che **non** permane alla terminazione del processo

b) la **dimensione di una pipe** è fissa
   ➔ ad una pipe è associato un buffer (p.e. di 4 kbytes)
   NB. L'**atomicità** delle **write** su una pipe è garantita solo per operazioni che non eccedono la dimensione di questo buffer

c) la pipe usa una gestione FIFO
   ➔ i primi dati scritti in una pipe sono i primi ad essere letti

Esempio

```
#include <stdio.h>
#define MSGSIZE 23
char *msg [10]= {
" Salve. Messaggio  #0"," Salve. Messaggio  #1",
" Salve. Messaggio  #2"," Salve. Messaggio  #3",
" Salve. Messaggio  #4"," Salve. Messaggio  #5",
" Salve. Messaggio  #6"," Salve. Messaggio  #7",
" Salve. Messaggio  #8"," Salve. Messaggio  #9" };

main ()
{
 int pid, j, piped[2], stato;
 char inpbuf [MSGSIZE];
 /* si crea una pipe */
 if(pipe(piped)<0 ) {perror("piped"); exit(1);}

 if((pid=fork())<0) {perror("fork"); exit (2);}

 if(pid != 0) {  /* padre */
    close(piped[0]); /*padre chiude lato lettura*/
    for(j=0; j<10; j++)
        write (piped[1], msg [j], MSGSIZE);
    wait(&stato);  exit(0);
 } else {  /* figlio */
    close(piped[1]);/*figlio chiude lato scrittura*/
    for (j=0; j<10; j++) {
        read(piped[0], inpbuf, MSGSIZE);
        printf("figlio legge :%s\n", inpbuf);
    }
    exit(0);
 }
}
```

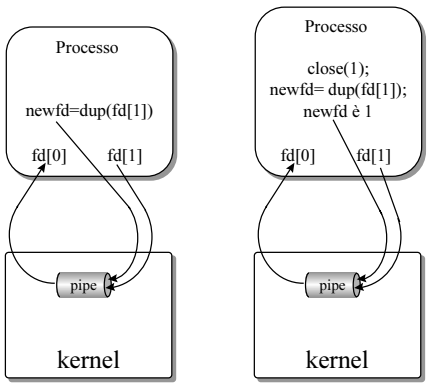

Altre primitive

DUP

```
#include <unistd.h>
int dup(fd);
int fd;
```

La dup ritorna un nuovo file descriptor che riferisce lo stesso file o la stessa pipe fornita come parametro (fd) (stesso file, stesso I/O pointer, stessi diritti).

Il nuovo file descriptor è quello con il numero più basso possibile nel sistema.



piping di due comandi

```
#include <stdio.h>

int join (com1, com2)
char *com1[], *com2[];
{
 int status;
 int pid;
 int piped[2];

 switch(fork()) {
   case -1: return (1); /* errore */
   case 0:  break; /*figlio */
   default: wait(&status);/*Padre attende figlio*/
            return(status>>8);
 }

 /* figlio esegue il comando: crea la pipe */
 if (pipe(piped)<0) {return(2);}

 /* creazione nipote */
 if((pid=fork())<0)  {return(3);}
   else
      if(pid!=0) {    /* Figlio */
         close(1);    /* chiusura stdout */
         dup(piped[1]); /*output sulla pipe */
         close(piped[0]); close(piped[1]);
         execvp(com1[0],com1);
         return(4); /* errore */
      } else { /* Nipote */
         close(0); /* input dalla pipe */
         dup (piped[0]);
         close(piped[0]); close(piped[1]);
         execvp(com2[0],com2);
         return(5); /*errore*/
}    }
```