La Comunicazione tra Processi in Unix

Ing. Cristiano Gregnanin - Dipartimento di Informatica

Interazione tra processi Unix

- I processi Unix **non** possono condividere memoria
- L'interazione tra processi può avvenire:
  - mediante la condivisione di file:
    - **complessità** : realizzazione della sincronizzazione tra i processi.
  - attraverso specifici strumenti di Inter Process Communication:
    - per la comunicazione tra processi sulla stessa macchina:
      - **pipe** (tra processi della stessa gerarchia)
      - fifo (qualunque insieme di processi)
    - per la comunicazione tra processi in nodi diversi della stessa rete:
      - **socket**

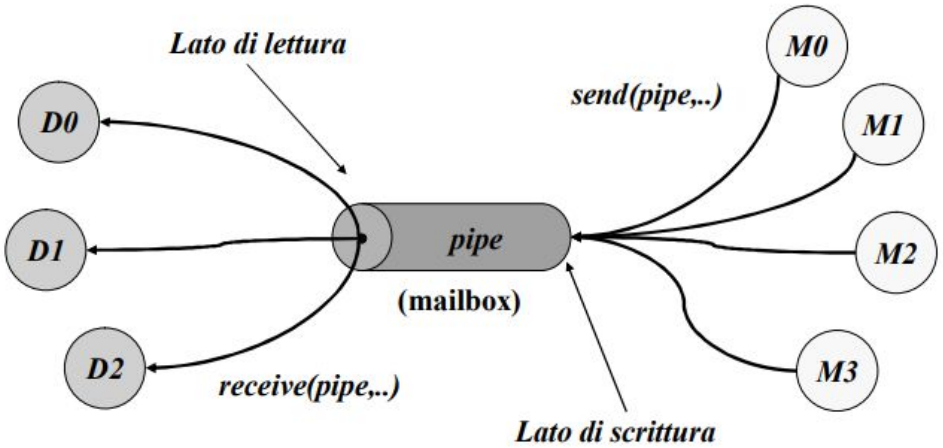
Pipe

La pipe è un canale di comunicazione tra processi:

- **unidirezionale**: accessibile ad un estremo in lettura ed all'altro in scrittura
- **molti-a-molti:**
  - più processi possono spedire messaggi attraverso la stessa pipe
  - più processi possono ricevere messaggi attraverso la stessa pipe
- **capacità limitata**:
  - la **pipe** è in grado di gestire **l'accodamento** di un numero limitato di messaggi, gestiti in modo **FIFO**: il limite è stabilito dalla **dimensione** della pipe (es.4096 bytes).

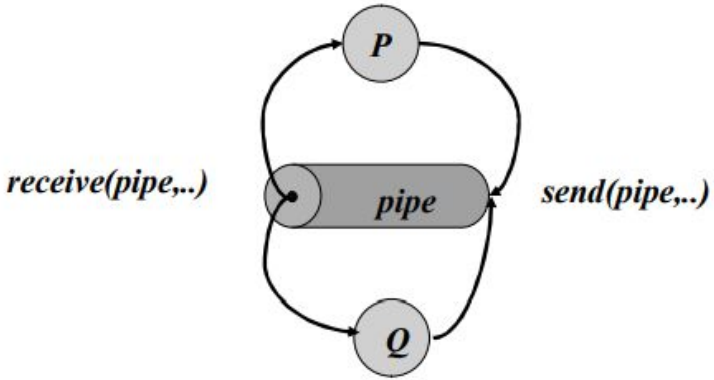
Comunicazione attraverso pipe

Mediante la pipe, la comunicazione tra processi è **indiretta** (senza naming esplicito). Esempio: cassetta delle lettere



Pipe: unidirezionalità/bidirezionalità

- Uno stesso processo può:
  - sia **depositare** messaggi nella pipe (**send**), mediante il lato di **scrittura**
  - che **prelevare** messaggi dalla pipe (**receive**), mediante il lato di **lettura**
- La pipe può anche consentire una **comunicazione "bidirezionale"** tra i processi P e Q (ma va rigidamente **disciplinata** !)



System call pipe

Firma della funzione pipe:

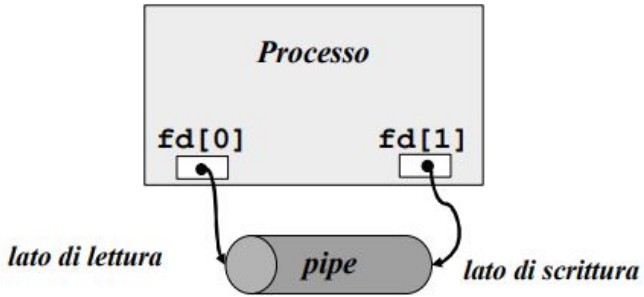
```
#include <unistd.h>;

int pipe(int fd[2]);
```

- **fd** è il puntatore a un vettore di **2 file descriptor**, che verranno inizializzati dalla system call in caso di successo:
  - fd[**0**] rappresenta il lato di **lettura** della pipe
  - fd[**1**] è il lato di **scrittura** della pipe
- la system call pipe **restituisce**:
  - un valore negativo, in caso di fallimento
  - **0, se ha successo**

Creazione di una pipe

Se **pipe(fd)** ha successo:

- vengono allocati due nuovi elementi nella tabella dei file aperti del processo e i rispettivi file descriptor vengono assegnati a fd[0] e fd[1]:
  - **fd[0]: lato di lettura (receive) della pipe**
  - **fd[1]: lato di scrittura (send) della pipe**



Omogeneità con i file

Ogni lato di accesso alla pipe è visto dal processo in modo omogeneo al file (file descriptor):

si può accedere alla pipe mediante le system call di accesso a file:

- ssize_t **read**(int fd, void *buf, size_t count);
  - La funzione **read()** legge da **fd** numero **count** bytes inserendoli nel buffer puntato da **buf**.
    - **buf** deve essere di dimensioni adeguate, cioè' almeno di **count** bytes.
    - La funzione **read()** ritorna il numero di elementi letti. In caso di raggiungimento della fine file viene restituito un numero minore di **count**. In caso di errore viene restituito -1.

- ssize_t **write**(int fd, const void *buf, size_t count);
  - La funzione **write()** scrive su **fd** numero **count** bytes di dati presenti nel buffer puntato da **buf**.
  - La funzione **write()** ritorna il numero di elementi scritti. In caso di errore viene restituito -1.

Comunicazione

Chiaramente creare una pipe all'interno di un singolo processo non serve a niente;

se però ricordiamo che **un processo figlio è sostanzialmente un clone del padre** è immediato capire come una pipe possa diventare un meccanismo di intercomunicazione, dato che **un processo figlio infatti condivide gli stessi file descriptor del padre, compresi quelli associati ad una pipe**

Sincronizzazione dei processi comunicanti

Il **canale** (la **pipe**) ha capacità limitata quindi è necessario sincronizzare i processi:

- se la pipe è **vuota**: un processo che **legge** si **blocca**
- se la pipe è **piena**: un processo che **scrive** si **blocca**

Sincronizzazione automatica: read e write da/verso pipe possono essere sospensive !

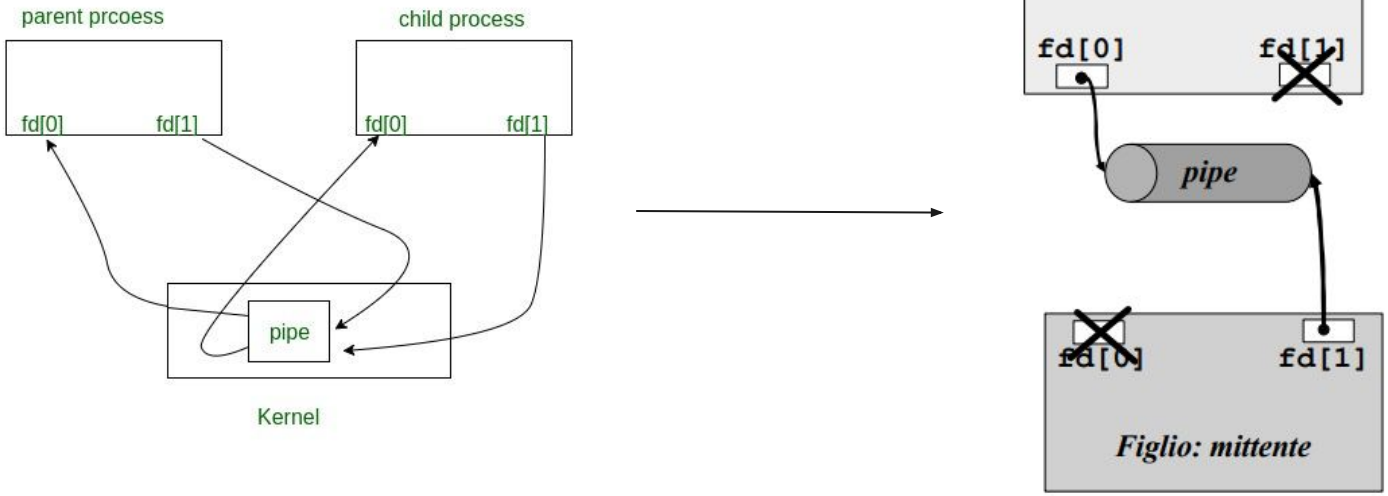
Quali processi possono comunicare mediante pipe?

Per mittente e destinatario il riferimento al canale di comunicazione è un file descriptor:

- Soltanto i **processi appartenenti a una stessa gerarchia** (cioè, che hanno un antenato in comune) possono **scambiarsi messaggi mediante pipe**;
- ad esempio, possibilità di comunicazione:
  - tra processi fratelli (che ereditano la pipe dal processo padre)
  - tra un processo padre e un processo figlio;
  - tra nonno e nipote
  - etc etc

Comunicazione tra padre e figlio

Ogni processo chiude il lato della pipe che non usa



Chiusura di pipe

- Ogni processo può chiudere un estremo della pipe con una **close**
- Un estremo della pipe viene effettivamente chiuso quando tutti i processi che ne avevano visibilità hanno compiuto una **close**
- Se un processo P:
  - tenta una **lettura** da una pipe il cui lato di scrittura è effettivamente chiuso: **read ritorna 0**
  - tenta una **scrittura** da una pipe il cui lato di lettura è effettivamente chiuso: **write ritorna -1**

Pipe bidirezionali

Per convenzione le pipe vengono utilizzate come canali di comunicazione unidirezionali.

Se due processi richiedono un canale di comunicazione bidirezionale tipicamente si creano due pipe, e si chiudono gli estremi non usati.

Esempio

```
int    fd[2], nbytes,pid;
char   string[] = "Hello, world!\n";  readbuffer[80];
pipe(fd);
if(pid=fork()== 0)      {
      close(fd[0]);
      write(fd[1], string, (strlen(string)+1));
      exit(0);
}
else {
      close(fd[1]);
      nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
      printf("Received string: %s", readbuffer);
   }
```