Esercitazione 5
Gruppo LZ

Comunicazione tra processi Unix: pipe

System Call relative alle pipe

`pipe`	• **`int pipe (int fd[])`** crea una pipe e assegna i 2 file descriptor relativi agli estremi di lettura/scrittura ai primi due elementi dell'array fd. • Restituisce 0 in caso di creazione con successo, -1 in caso di errore
`close`	• Stessa system call usata per chiudere file descriptor di file regolari • Nel caso di pipe, usata da un processo per chiudere l'estremità della pipe che non usa.

Primitive di comunicazione

`read`	• Stessa system call usata per leggere file regolari, ma può essere bloccante: Se la pipe è vuota: il processo chiamante attende fino a quando non ci sono dati disponibili.
`write`	• Stessa system call usata per scrivere su file regolari, ma può essere bloccante: Se la pipe è piena: il processo chiamante attende fino a quando non c'è spazio sufficiente per scrivere il messaggio.
`dup`	fd1=dup(fd) crea una copia dell'elemento della tabella dei file aperti di indice fd. • La copia viene messa nella prima posizione libera (in ordine crescente di indice) della tabella dei file aperti. • Assegna a fd1 l'indice della nuova copia, -1 in caso di errore

Esercitazione 5 - Obiettivi

- Esercizio 1: utilizzo di system call relative a
  - file
  - pipe
- Esercizio 2: redirezione di input e output

- Ai fini del bonus è possibile consegnare uno qualsiasi dei due esercizi

Esercizio 1 (1/2)

Si realizzi un programma di sistema in C che preveda la seguente interfaccia:

```
./inverti_e_filtra Fin Car
```

dove:

- **`Fin`** è il nome (assoluto o relativo) di un file di testo esistente nel file system.
- **`Car`** è un carattere

Il processo P0 crea due figli.

P1 legge il contenuto di **`Fin`** dalla fine all'inizio e lo invia a P2.

P2 legge quanto inviatogli da P1 e stampa sullo standard output le sole line che iniziano per **`Car`**.

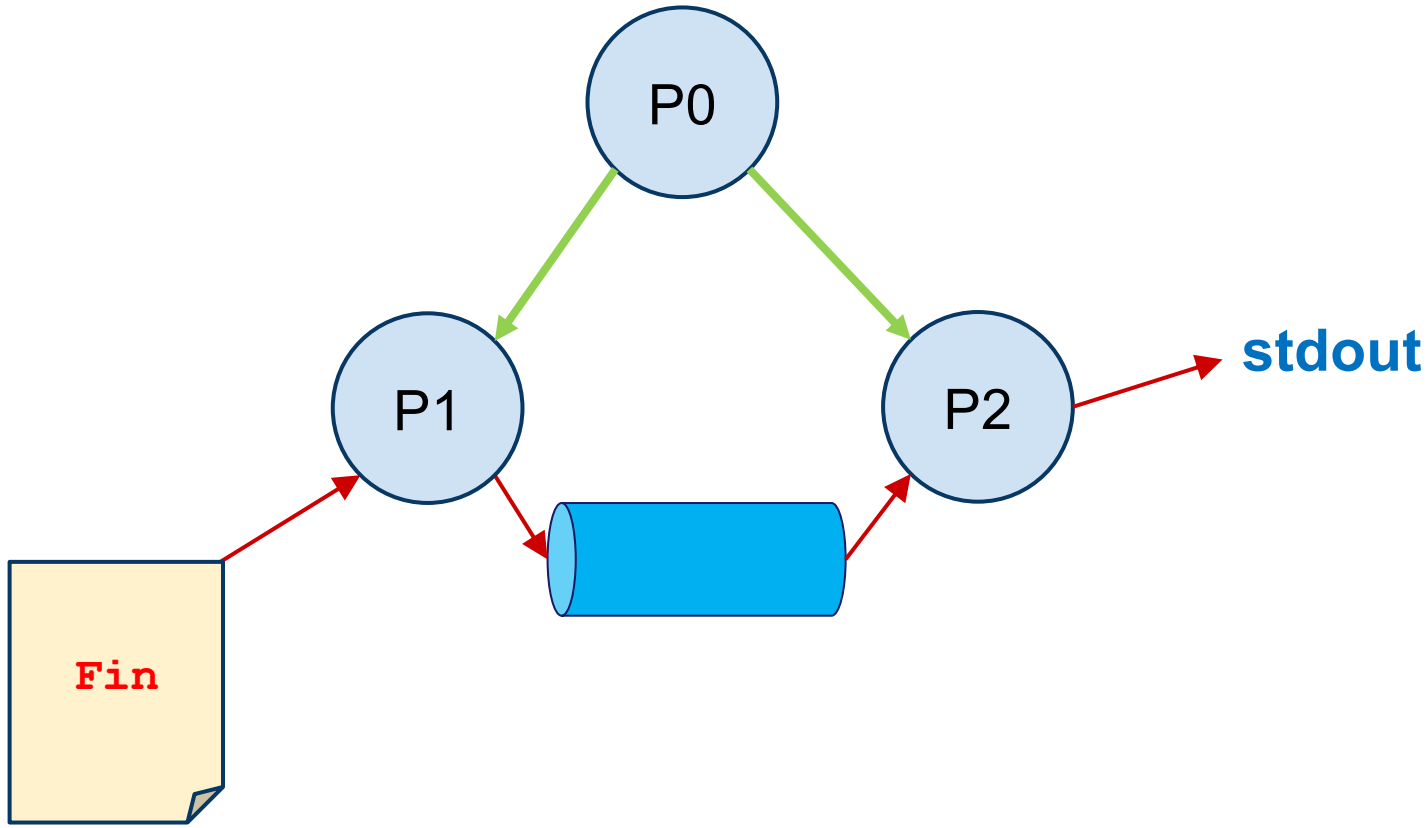
P0 attende la terminazione dei figli e ne raccoglie lo stato

Esercizio 1 - Riflessioni

- «P1 legge il contenuto di `Fin` dalla fine all'inizio» Come?
  Occorre una system call per saltare indietro ad ogni lettura
  NB: ricordare che ad ogni lettura l'IO pointer viene spostato sul byte successivo all'ultimo letto.

- **Come realizzare la comunicazione tra P1 e P2?**
  -> creazione di una pipe
- P1 non conosce il pid di P2… è un problema?

Modello di soluzione



Pipe - Riflessioni

Le pipe sono uno strumento di comunicazione tra processi

- Consentono a processi in gerarchia di scambiarsi dati

Alcune differenze rispetto a read-write su file:

- **`read`** e **`write`** bloccanti (se la pipe è rispet. vuota o piena)
- La **`read`** ritorna zero se e solo se tutti i fd relativi al lato di scrittura sono chiusi

⇒Perché è importante NON LASCIARE APERTE ESTREMITA' INUTILIZZATE DELLE PIPE?

Le pipe possono essere uno strumento di sincronizzazione tra processi. Quando conviene usare pipe e quando segnali?

- Se devo comunicare dei dati tra processi, sono più comode le pipe,
- ma se un processo deve fare delle operazioni intanto che aspetta di ricevere qualcosa da un altro, devo ricorrere ai segnali!

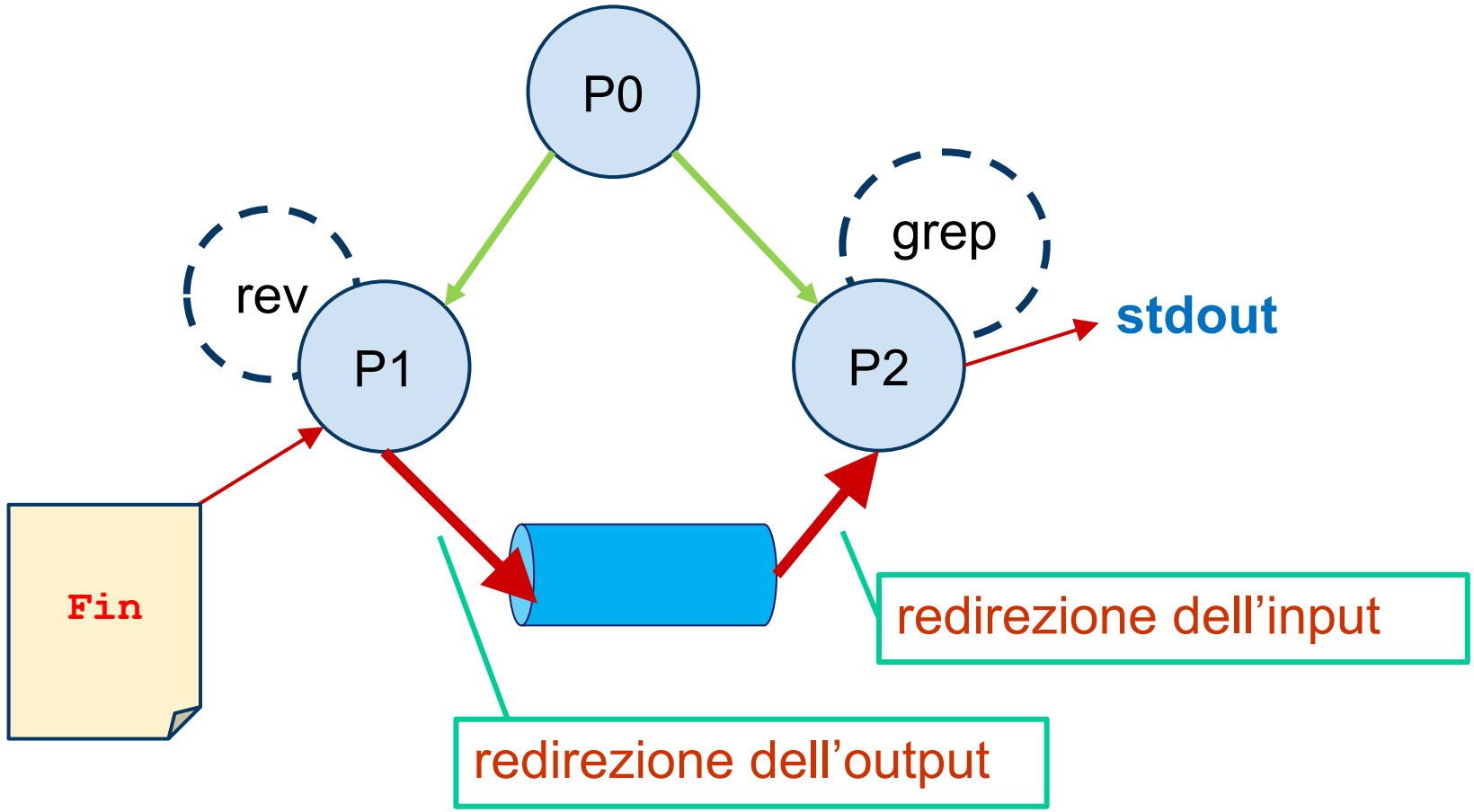
Esercizio 2 – redirezione comandi

Come esercizio 1, ma:

- Il figlio P1 realizza la lettura dalla fine all'inizio con il comando **`rev`**
- P2 filtra le righe ricevute da P1 che iniziano per **Car** usando il comando **`grep`**.
  - Esempio: **`grep ^p file1`**

    filtra tutte le righe di file1 che iniziano per il carattere p.

Modello di soluzione

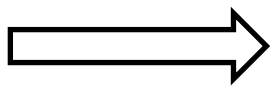


Esercizio 2 - Note

- Comando **`rev [file]`**
- Inverte l'ordine dei caratteri di ogni linea di file

FileDiEsempio

```
Questo è
il contenuto
del file
```



rev FileDiEsempio:

```
è otseuQ
otunetnoc li
elif led
```

Pertanto, il risultato di rev è leggermente diverso dall'inversione «totale» del contenuto di FileName richiesta nell'esercizio 1. Ai fini dell'esercitazione non ci curiamo di questo dettaglio.