Esercitazione 5
Gruppo AK

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

Esercizio 1 (1/3)

Si realizzi un programma di sistema in C che effettua un'analisi a campione dei caratteri contenuti in due file di testo

Il programma deve prevedere la seguente sintassi di invocazione:

```
./analisi Fa Fb
```

- **`Fa`** ed **`Fb`** sono nomi assoluti di file **di testo** esistenti nel file system

Si assuma che **`Fa`** ed **`Fb`**:

- non contengano nessun **`'\n'`**
- abbiano la stessa lunghezza

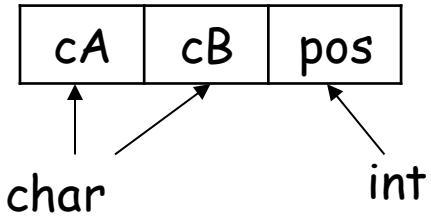
Esercizio 1 (2/3)

Il processo P0 deve generare un unico figlio P1.

Poi P0 legge randomicamente e confronta i due file **`Fa`** ed **`Fb`** dall'inizio alla fine, ovvero esegue continuamente i seguenti step:

- genera un numero **r** tra 0 e 4.
- sposta i due I/O pointer avanti di **r** posizioni
- legge il carattere corrispondente da **`Fa`** e da **`Fb`**
- confronta i due caratteri letti

Per ogni differenza riscontrata, P0 deve **inviare a P1** due caratteri (diversi) letti e la loro posizione



Esercizio 1 (3/3)

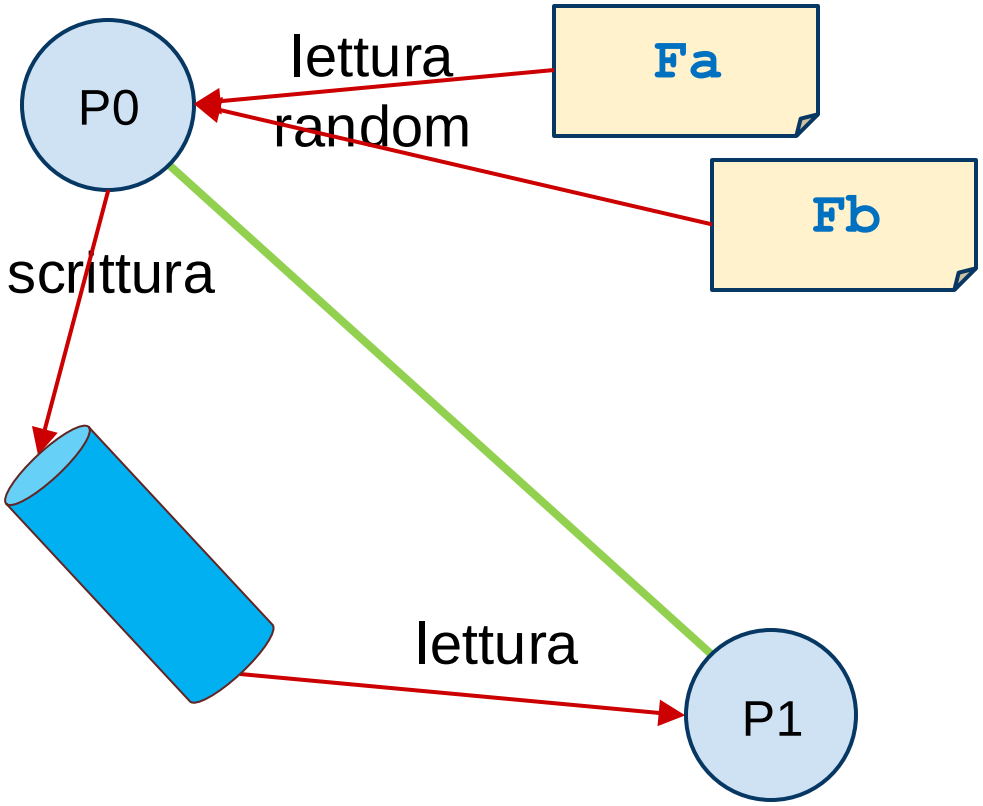
Il processo P1 deve leggere le triplette <***cA,cB,pos***> inviategli da P0 e per ognuna deve stampare a video un messaggio dei tipo:

*«P1: Trovata una differenza in posizione **pos** :: cA=**cA** cB=**cB**»*

Una volta lette tutte le triplette, il processo P1 deve stampare a video il messaggio

«P1: Lettura da pipe terminata»

Modello di soluzione



*<<P1: Trovata una differenza in posizione **pos** :: cA=**cA** cB=**cB**>>*

...

<<P1: Lettura da pipe terminata>>

Esercizio 1 – Riflessioni(1/2)

P0 deve **inviare** a P1 i caratteri (diversi) e le loro posizioni

➔ Quale strumento?

Tale «invio» è realizzabile in due modi (entrambi validi):

- scrittura/lettura di due char e poi di un int
- scrittura/lettura di una struttura:

```
typedef struct{
    char cA; //carattere letto da Fa
    char cB; //carattere letto da Fb
    int pos; //posizione
}elemento;
```

Esercizio 1 – Riflessioni(2/2)

Una volta lette tutte le triplette, il processo P1 deve stampare a video il messaggio *«P1: Lettura da pipe terminata»*

➔ Come può P1 sapere quando sono finite le triplette?

RICORDARE la differenza tra read-write su file e read-write su pipe:

- **`read`** e **`write`** su pipe sono bloccanti (se la pipe è rispet. vuota o piena)
- In particolare la read si sblocca:
  - Quando c'è qualcosa da leggere
  - Quando risultano chiusi tutti i fd relativi ai lati di scrittura della pipe (read ritorna 0)

  ➔Pertanto, è importante NON LASCIARE MAI APERTE ESTREMITA' INUTILIZZATE DELLE PIPE!

Pipe – Riflessioni generali

Le pipe sono uno strumento di comunicazione tra processi

- Consentono a processi in gerarchia di scambiarsi dati

Le pipe possono anche 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 (1/3)

Si realizzi una variante dell'esercizio 1 in cui:

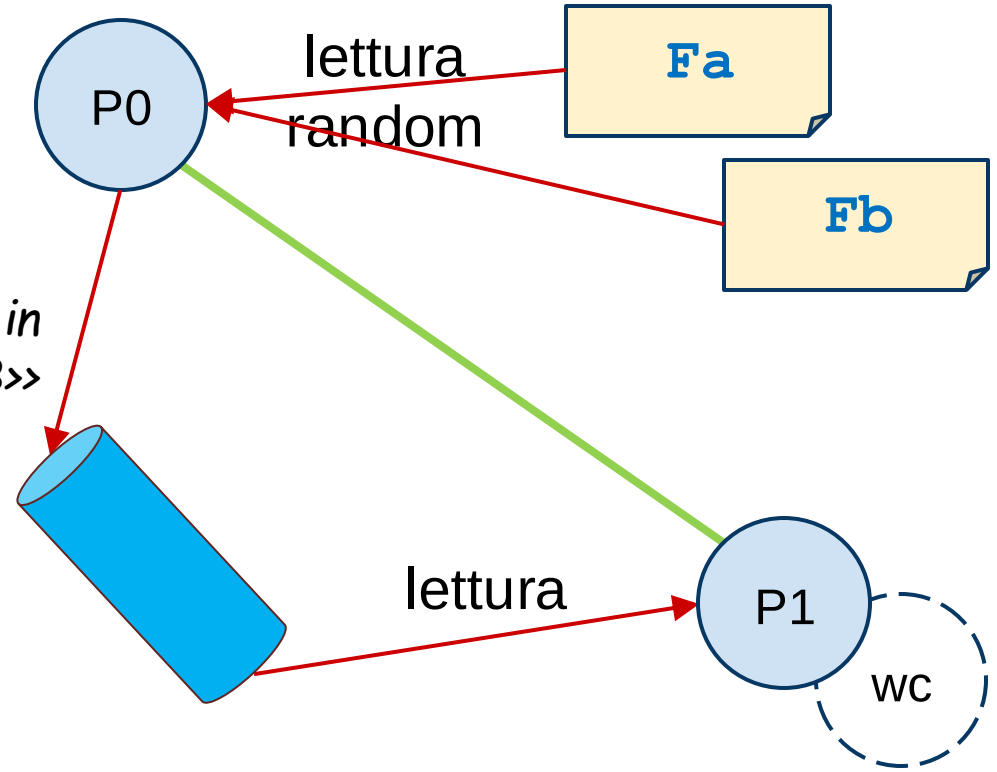
Per ogni differenza riscontrata, P0 deve stampare sullo stdout e inviare a P1 una riga (terminata da '\n') con il seguente messaggio:

*«P0: Trovata una differenza in posizione **pos** :: cA=**cA** cB=**cB**»*

P1 deve **contare** le differenze trovate utilizzando il comando di shell **`wc`**

RICORDARE: wc è un comando di shell che permette di contare (a seconda dell'opzione passata) i caratteri, le parole o le linee in un file

Modello di soluzione



scrittura di
*<<P0: Trovata una differenza in posizione **pos** :: cA=**cA** cB=**cB**>>*

stampa sullo stdout il numero N di linee lette da pipe = numero di differenze trovate

Esercizio 2 – Riflessioni

Abbiamo usato wc per contare i caratteri, le parole o le linee da un file, ma il man di wc riporta:

SYNOPSIS

- **wc** [**-clw**] [file ...]

Il parametro "file" è tra […] ➔ è opzionale

Se non viene passato alcun parametro wc legge da stdin

L'esercizio richiede di leggere da pipe ➔ occorrerà redirigere opportunamente lo stdin