

Quinta Esercitazione

Comunicazione tra
processi Unix: pipe

Agenda

Esercizio 1 - da svolgere

Scambio di informazioni tra processi tramite pipe

Esercizio 2 - da svolgere

Redirezione di **stdin** e **stdout** su pipe

Esercizio 3 - da svolgere

Elaborazione di dati in "pipeline"

Primitive di comunicazione

pipe	<ul style="list-style-type: none">• 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
dup	<ul style="list-style-type: none">• fd1=dup(fd) crea una copia del descrittore fd.• La copia viene messa nella prima posizione libera della tabella dei file aperti.• Assegna a fd1 la nuova copia del descrittore, -1 in caso di errore

Primitive di comunicazione

read	<ul style="list-style-type: none">• Stessa system call usata per leggere file regolari, ma... Bloccante!• Se la pipe è vuota il processo chiamante si blocca fin quando non ci sono dati disponibili.
write	<ul style="list-style-type: none">• Stessa system call usata per scrivere su file regolari, ma... Bloccante!• Se la pipe è piena il processo chiamante si blocca fin quando non c'è spazio per scrivere.
close	<ul style="list-style-type: none">• 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 gli interessa


Esercizio 1 - IPC e sincronizzazione tramite pipe

- Si realizzi un programma C che, usando le opportune system call unix, realizzi la seguente interfaccia:

./correggi file_in file_out

- **file_out**: nome di un file non esistente nel filesystem
- **file_in**: nome di un file binario esistente contenente N triplette di numeri interi, con N non noto a priori.

A	B	C	A	B	C	A	B	C	
1	3	3	-53	-2	-2	12	-1	12	...



- NB: file binario ≠ file di testo
-

Esercizio 1 - Traccia (2/3)

Il programma deve realizzare il seguente comportamento:

- Il processo padre (P0) deve generare due figli P1 e P2
 - Il processo P2 deve:
 - Leggere i primi due interi (A,B) di ogni tripletta in `file_in`;
 - Al termine della lettura dei primi due elementi di ogni tripletta, P2 deve comunicare a P1 il valore del maggiore;
 - Letta l'ultima tripletta, segnalare a P1 il termine della sua elaborazione.
-

Esercizio 1 - Traccia (3/3)

Il processo **P1** deve

- Leggere il valore di *C* e, nel caso in cui questo risultasse diverso dal massimo comunicatogli da *P2*
- Scrivere il valore dell'intero maggiore al posto del relativo elemento *C* della tripletta
- **Comunicare a *P0* il valore** corretto

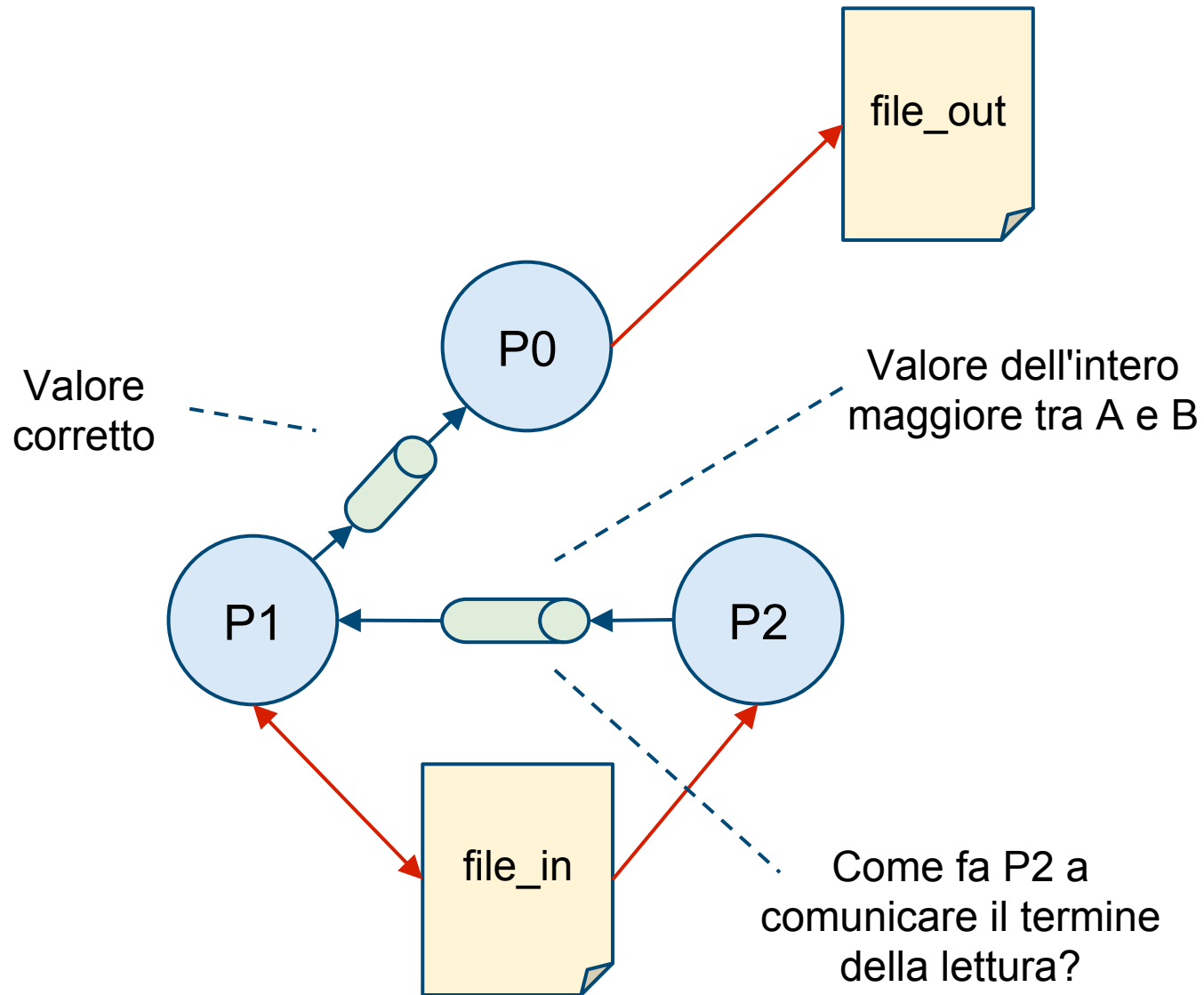
Il processo **P0** deve

- **Sommare i valori ricevuti da *P1***
 - Al termine dell'elaborazione dei figli, scrivere tale valore su `file_out`
-

Esercizio 1 - Note alla soluzione

- Uso di pipe vs uso di segnali per sincronizzare processi
 - **Quante pipe** occorre creare?
 - Comunicazione tra P2 e P1
 - Comunicazione tra P1 e P0
 - **Quando crearle?**
 - Dipenderà da chi le deve usare...
 - La pipe tra P0 e P1 può esser creata in P1?!
 - E quella tra P1 e P2?
-

Esercizio 1 - Modello di soluzione



Esercizio 1 - Note alla soluzione

- **Chiudere subito** gli estremi che non mi interessano delle pipe!
 - Come fa **P2** a **comunicare a P1 il termine** della sua elaborazione?
 - Un **segnale di fine**, o è possibile farlo anche con altri strumenti?
 - Chiusura lato scrittura pipe?
-

Pipe - Riflessioni post-esercizio

Le pipe sono uno strumento di **comunicazione** tra processi

- Consentono a processi in gerarchia di scambiarsi dati

Alcune differenze

- **read e write bloccanti** (se la pipe è rispettivamente vuota o piena)
- Una **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!
-

Digressione: atomicità delle operazioni su pipe

Scenario:

Due processi scrivono in maniera concorrente su una stessa pipe.

L'ordine è non deterministico
(dipende dalle scelte di scheduling)

Domanda:

la write è una operazione atomica?

Digressione: atomicità delle operazioni su pipe

```
// Inizializzazione dei buffer ...  
char large1[BIG_NUMBER], large2[BIG_NUMBER];  
// da qualche parte nel codice di P1  
write(fdpipe[1], large1, BIGNUMBER);  
// da qualche parte nel codice di P2  
write(fdpipe[1], large2, BIGNUMBER);
```

chi mi assicura che il risultato sia questo?



E se invece fosse questo?!



Digressione: atomicità delle operazioni su pipe

POSIX.1-2001 prevede che scritture fino a 512 byte siano atomiche

Linux 2.6.11 garantisce che scritture fino a 65 Kilobyte siano atomiche

Per il resto è possibile (e probabile) che le scritture di più processi concorrenti siano **interleaved** sulla pipe, con risultati spesso spiacevoli per i processi che devono ricevere i dati

...\$ **man pipe**

Esercizio 2 - Redirezione di `stdin` e `stdout` su pipe

Si realizzi un programma C che, utilizzando le system call di Unix, abbia interfaccia di invocazione:

```
./cerca file1 file2 C
```

- `file1` e `file2` : path di file di testo esistenti nel file system. Ciascuno ha un numero variabile di righe, ognuna delle quali ha lunghezza massima di 100 caratteri.
 - `C`: un carattere
-

Esercizio 2 (2/3)

Il processo padre **P0** genera due figli (P1 e P2)

P1 deve:

- Leggere **file1** e contare il numero **M** di righe in cui **c** compare almeno una volta
- Ricevere da **P2** il numero **N** di righe contenenti **c** in **file2**
- Una volta ottenuti sia **M** che **N**, comunicarli a **P0**

P2 deve:

- Contare in **file2** il numero **N** di righe in cui **c** compare almeno una volta tramite il comando **grep**
- Comunicare tale valore a **P1** tramite opportuna redirectione dello **stdout** di **grep**

P0 deve:

- Stampare a video il numero totale di righe con occorrenze di **c** in **file1** e **file2**

Esercizio 2 - Suggerimenti

P2 deve contare tramite l'esecuzione del comando **grep**

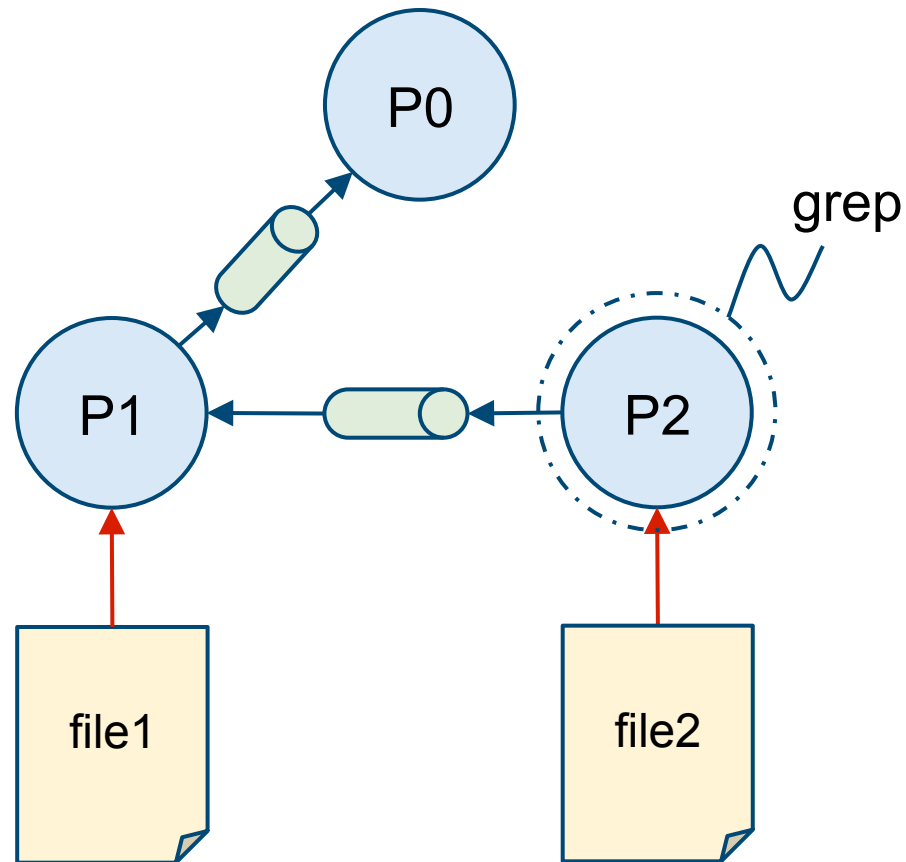
man grep

grep -c C file2

Attenzione:

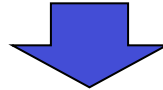
L'output di **grep -c C** è una stringa che rappresenta il numero di occorrenze di **C**, non è di tipo binario!

Esercizio 2 - Modello di soluzione

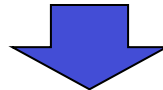


Esercizio 2 - Variante

Si supponga di non poter usare l'opzione `-c` per `grep`



- `grep C file2` stampa in output le righe di `file2` contenenti `C`
- `wc -l` conta il numero di linee nel suo standard input



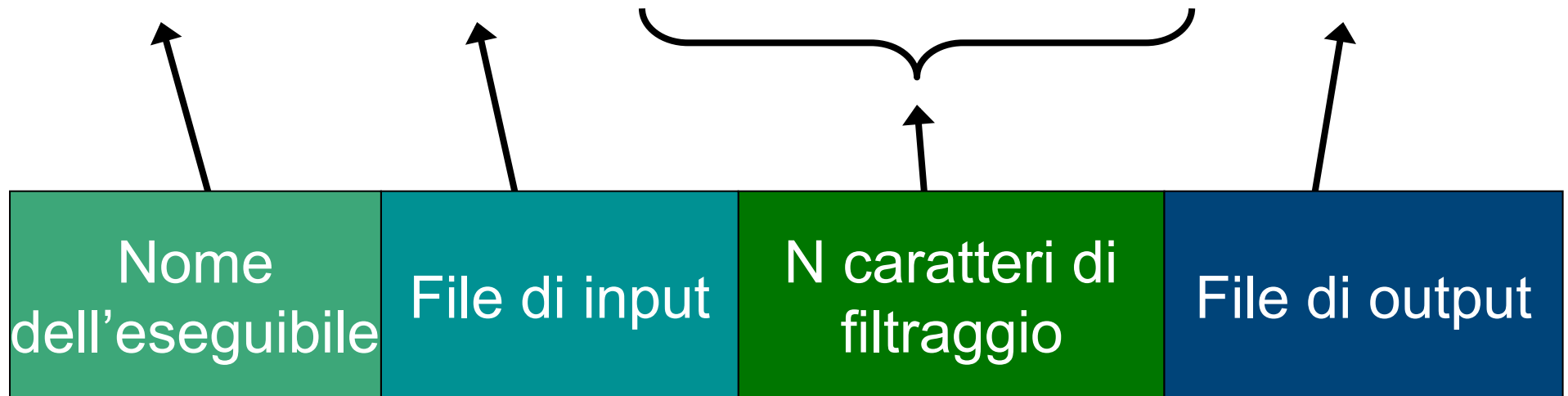
Idea: piping di `grep` e `wc`

- Un ulteriore fratello `P3` lancia `grep` in pipe verso `P2`
 - `P2` esegue il comando `wc`
-

Esercizio 3 - Elaborazione dati in pipeline

Scrivere un comando che abbia la sintassi:

filter <input> <car1> ... <carN> <output>



Il comando deve scrivere nel file di output il contenuto del file di input, filtrato con i caratteri dati

Esercizio 3 - Specifiche

In particolare:

- **il processo padre P0 crea N figli**, tanti quanti i caratteri passati da riga di comando (si faccia l'ipotesi esemplificativa di al massimo MAX figli, con MAX costante).
- **ogni figlio i-esimo P_i è associato all'i-esimo carattere car_i** : in particolare, P_i legge i dati fornitigli dal fratello P_{i-1} e trasferisce al fratello P_{i+1} i caratteri ricevuti, TOLTE tutte le occorrenze di car_i .

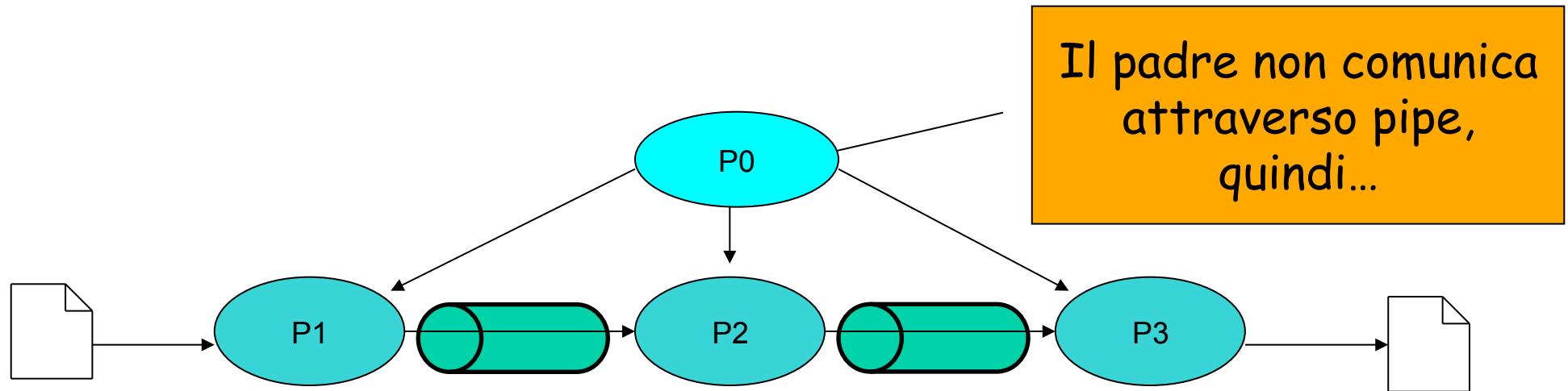
Ogni figlio legge i dati prodotti dal figlio precedente, li "**filtra**", e comunica il risultato al figlio successivo.

NB:

Il **primo figlio** legge dal file di input passato come argomento
L' **ultimo figlio** scrive sul file di output passato come argomento

Suggerimenti (1/2)

- Uso di pipe tra processi fratelli. Chi le crea?
- **N figli => N-1 pipe**
- ogni figlio utilizzerà solo 2 lati di due pipe (fatta eccezione del primo e ultimo figlio)
- Ogni processo, **COME PRIMA COSA**, dovrà **chiudere TUTTI i descrittori che non gli competono**



Suggerimenti (2/2)

- Il codice dei figli è uguale, fatta eccezione per
 - Il carattere di filtraggio (l'i-esimo figlio è associato all'i-esimo carattere)
 - Quali descrittori vanno chiusi? (dipende dalla posizione del figlio)
- Ad esempio, si può **definire una struttura dati** di descrittori del tipo:

Input_file	Pipe1[0]	...	PipeN-1[0]	X
X	Pipe1[1]	...	PipeN-1[1]	Output_file

Descrittori di lettura
Descrittori di scrittura

Input_file	Pipe1[0]	...	PipeN-1[0]	X
X	Pipe1[1]	...	PipeN-1[1]	Output_file

Esempio:
primo figlio P1