La comunicazione fra processi

dr. Andrea E. Naimoli

Università degli Studi di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione

via Sommarive 14

I - 38050 Trento - Povo, Italy

La comunicazione fra processi: concetti generali e piping

dr. Andrea E. Naimoli

Università degli Studi di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione

via Sommarive 14

I - 38050 Trento - Povo, Italy

Comunicazione tradizionale

- Il primo meccanismo di comunicazione tradizionale introdotto nei sistemi Unix sono le *pipe*
- Il suo utilizzo nei sistemi *unix-like* è esteso e quotidiano

ps | grep | sort

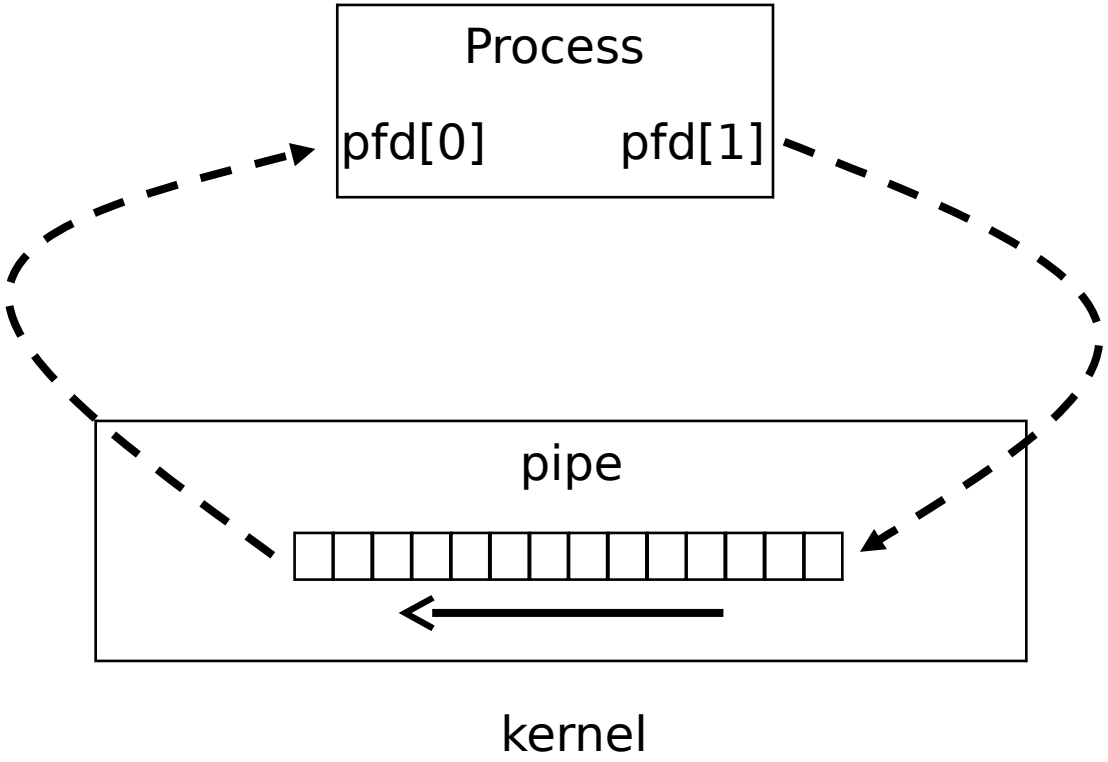
pipe

#include <unistd.h>

int pipe(int pfd[2]);

- Crea un canale di comunicazione rappresentato da 2 file descriptors
  - **pfd[0]** viene aperto in lettura
  - **pfd[1]** viene aperto in scrittura
- Gli *fd* non sono connessi a nessun file reale, ma a un buffer nel kernel

Schema di funzionamento



Dimensione e limite

- $ ulimit -a | grep pipe
  - ✓ pipe size (512 bytes, -p) 8 ->512 x 8 = 4096 byte
- Definito in
  - ✓ /usr/src/linux/include/linux/limits.h

write su pipe

int write (pfd[1], void *buf, int nbytes)

- I dati sono scritti in ordine di arrivo
- Quando la pipe è piena write si blocca finché read libera abbastanza spazio
- L'intero ritornato è il numero di byte effettivamente scritti sul file (-1 in caso di errore)

read su pipe

`int read (pfd[0], void *buf, int nbytes)`

- Se ci sono dati, vengono letti nell'ordine in cui sono stati scritti e una volta letti vengono cancellati dalla pipe (non possono essere riletti)
- Quando finiscono i dati (o si raggiunge **`nbytes`**) la read ritorna il numero di byte letti;
- Se la pipe è vuota la lettura si blocca finché arriva qualcosa da leggere.
- Se la pipe è chiusa in scrittura, **`read`** ritorna 0 (EOF)
- L'intero ritornato è il numero di byte effettivamente letti dal file (0 se EOF, -1 in caso di errore)

close su pipe

```
close(pfd[0]);  close(pfd[1]);
```

- Chiudendo il file descriptor si provoca un EOF
- Se si prova a scrivere su una pipe chiusa in lettura, **write** ritorna -1 per segnalare un errore

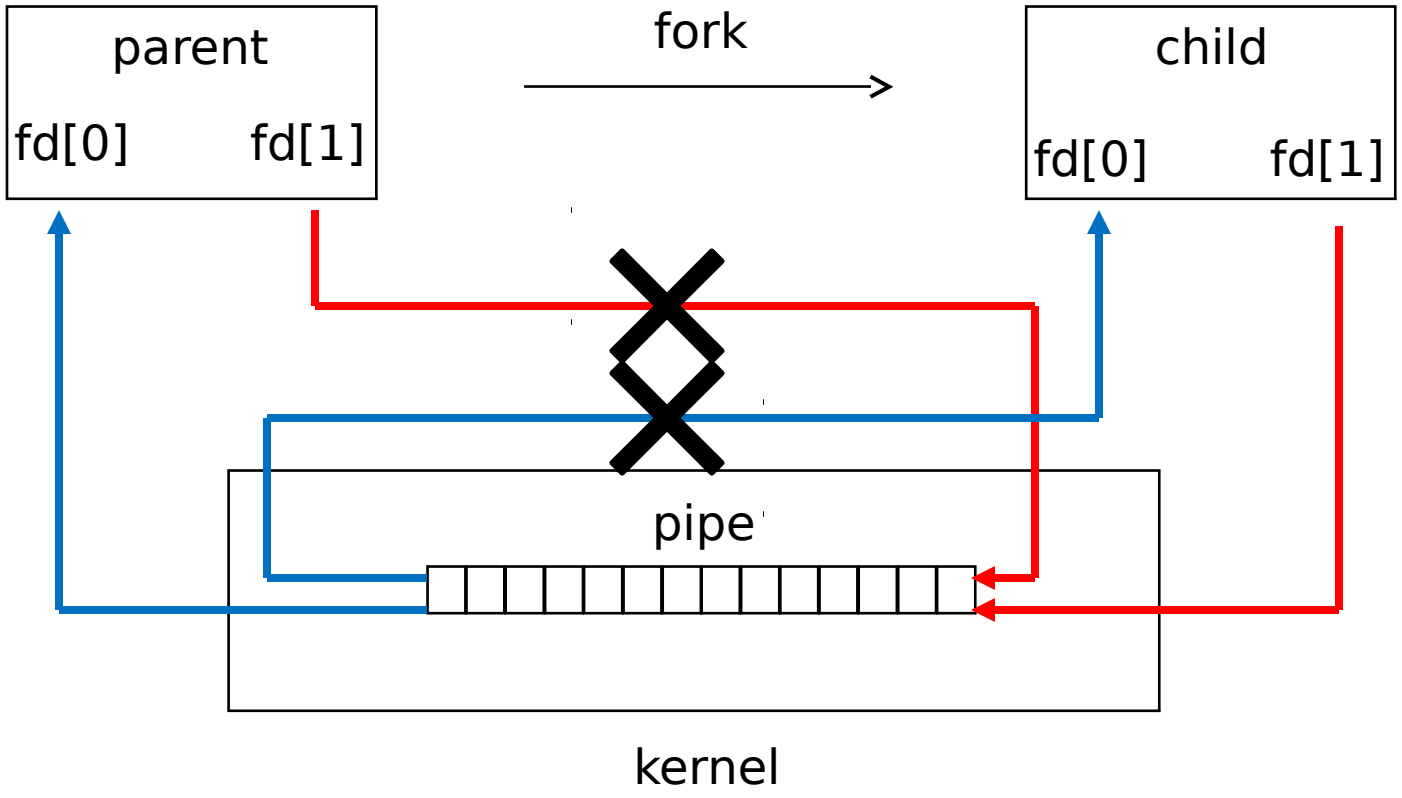
(Esempio “01-pipe.c”)

Comunicare con le pipe

- Forma più vecchia di *Inter Process Communication* in UNIX
- Sono half-duplex: i dati possono scorrere solo in una direzione
- Possono essere usate SOLO fra processi che hanno un antenato comune
- Normalmente una pipe è creata dal processo padre il quale poi chiama la **fork**
- La pipe viene dunque *copiata* dal padre al figlio
- In questo modo se uno dei due processi scrive su un capo della pipe, l'altro può leggere

(Esempio "02-pipe-fork.c")

Comunicare con le pipe



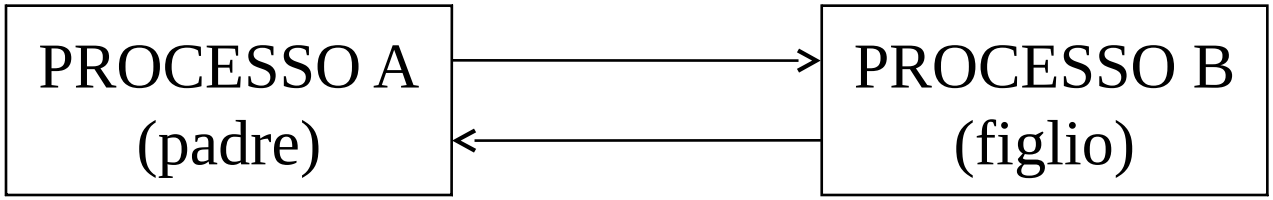
pipe - esempio

Voglio scrivere un programma che si comporti come la riga di comando **`who|wc`** (che permette di contare gli utenti collegati)

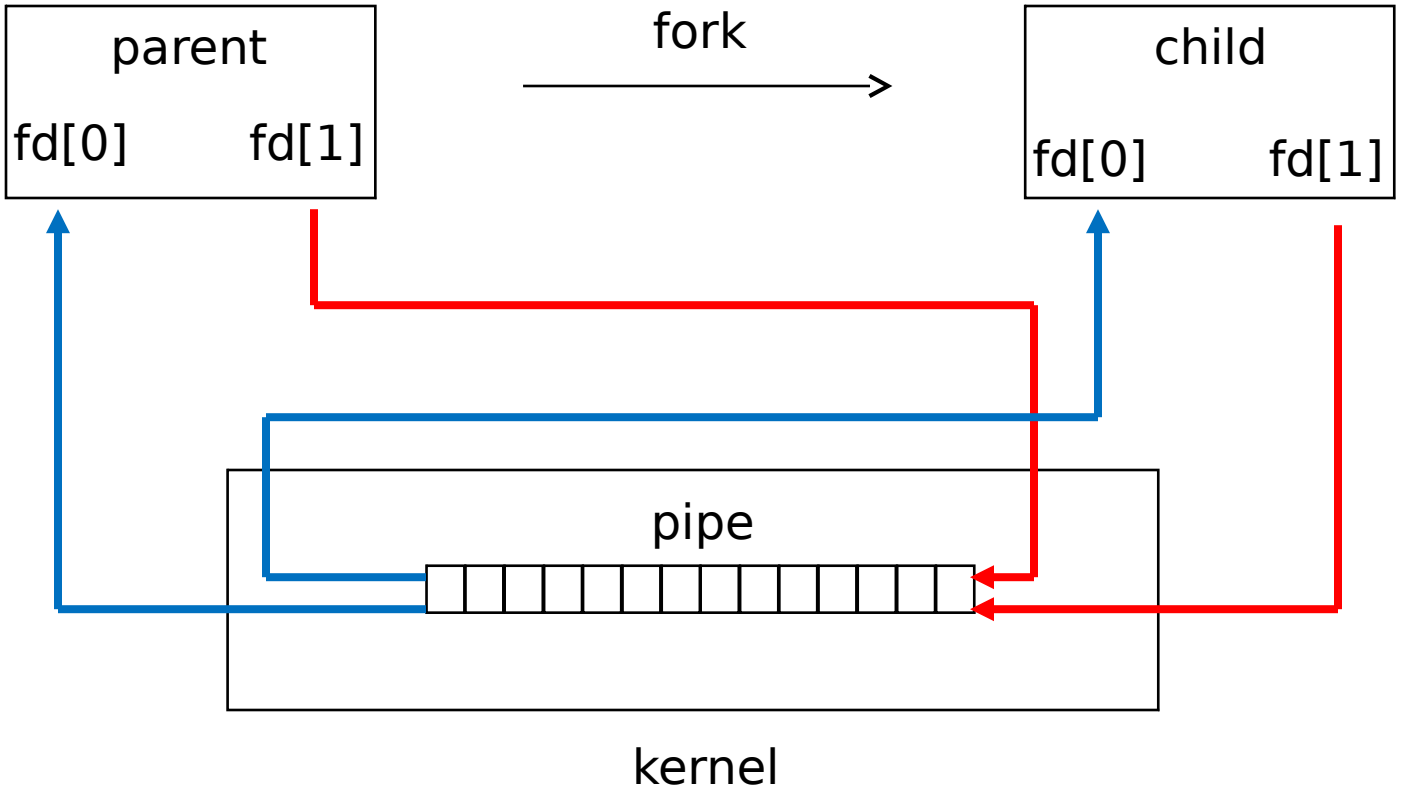
- Creare una pipe
- Creare un processo figlio
- Collegare lo *stdout* del figlio al *write* della pipe
- Collegare lo *stdin* del padre al *read* della pipe
- Il figlio esegue **`who`**
- Il padre esegue **`wc`**

`(Esempio “03-whowc.c”)`

Comunicazione full-duplex tra processi padre-figlio



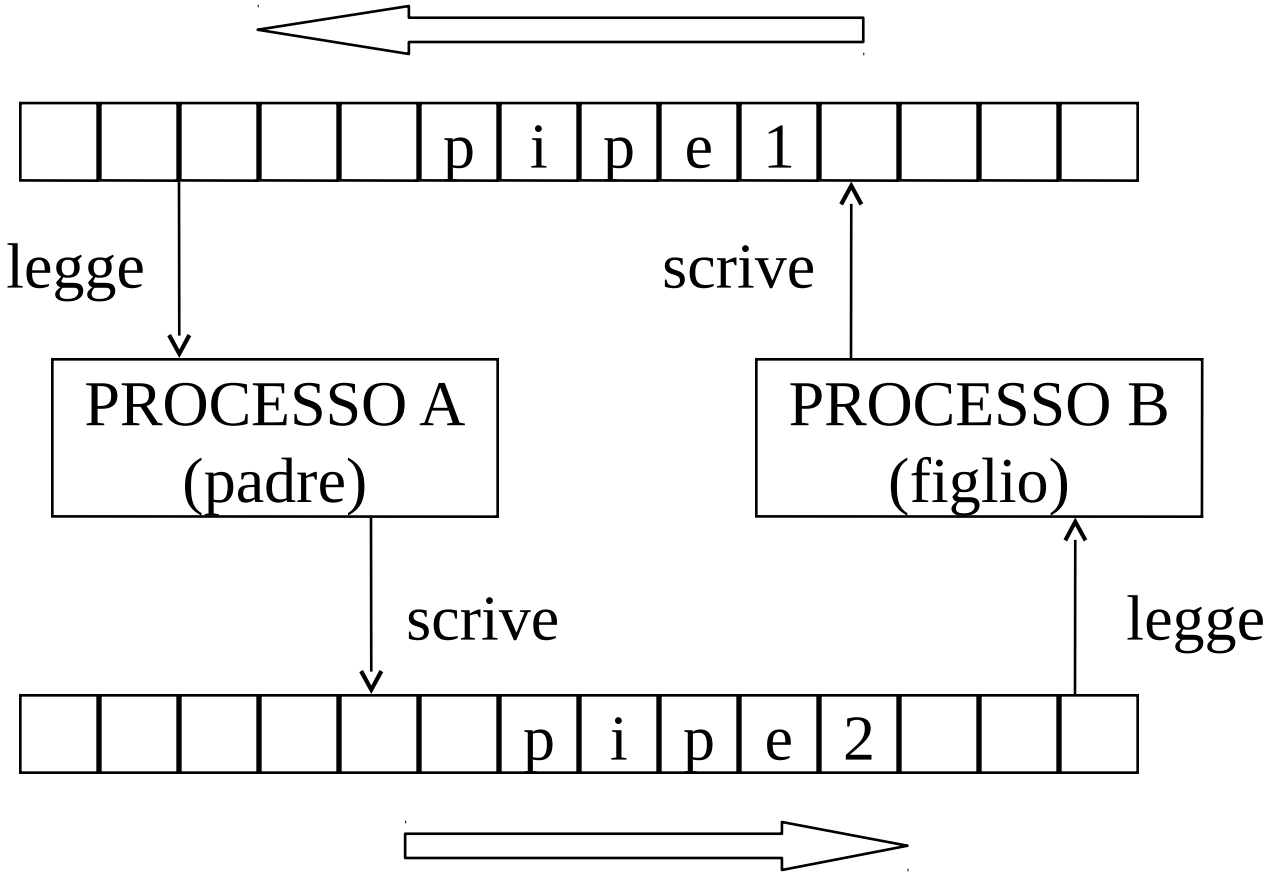
La soluzione sbagliata



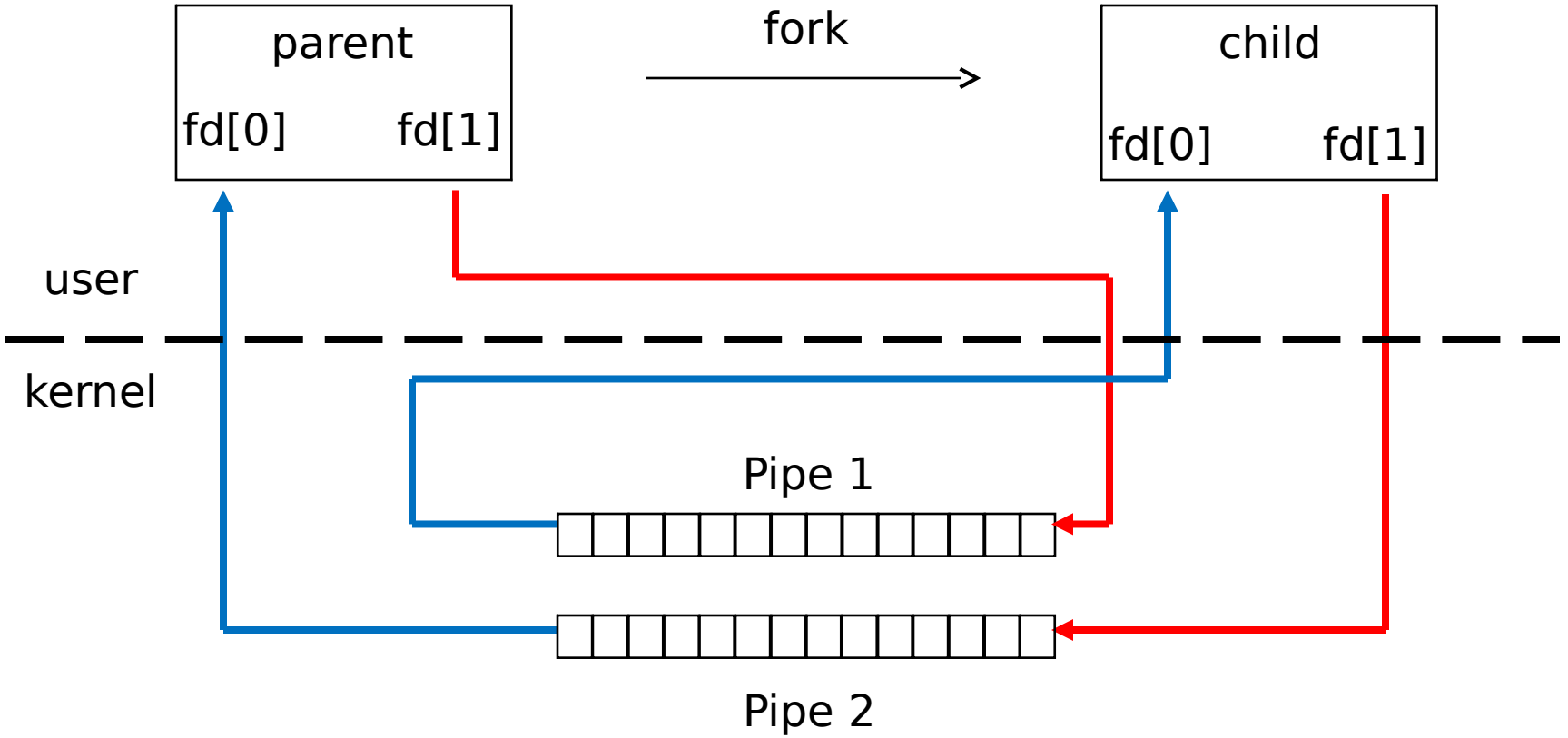
Cosa c’è che non va?

- I byte scritti dal padre non sono distinguibili da quelli scritti dal figlio
- In queste condizioni il padre legge tutto quello che trova, compreso quanto ha scritto lui
- Si può creare una situazione di stallo

La soluzione corretta



Implementazione



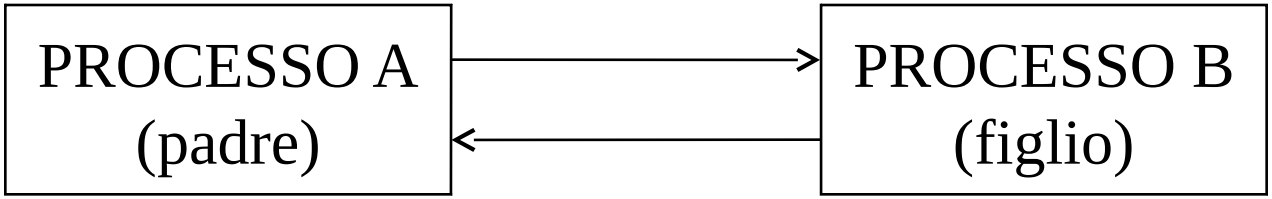
Esercizio (20 min.)

Scrivere un programma che permette una comunicazione full-duplex tra processo padre e processo figlio

(Esempio “04-pipe-full-duplex.c”)

Un problema delle pipe

- Possono comunicare attraverso le pipe soltanto processi con un antenato comune che deve essersi preoccupato di aprirle.



- In alcuni casi questo è limitativo. Occorre utilizzare uno strumento diverso: le **FIFO**

Esempio di applicazione

- minishell in grado di interpretare comandi con parametri dando significato corretto ai caratteri speciali:
  - & background
  - >, < ridirezione
  - | pipe [singole/multiple]

Bibliografia

- *GaPiL. Guida alla Programmazione in Linux*.. Simone Piccardi
- *Unix. Network Programming. Interprocess Communications*. Richard Stevens
- Operating System: Design and Implementation 3/e by Andrew Tanenbaum and Albert S. Woodhull
- *Modern Operating Systems*. Andrew. S. Tanenbaum, Prentice-Hall 1992, Edizione italiana *Moderni sistemi operativi*, Jackson Libri.