Modulo 9
Pipe e FIFO

Laboratorio di Sistemi Operativi I
Anno Accademico 2007-2008

Francesco Pedullà
(Tecnologie Informatiche)

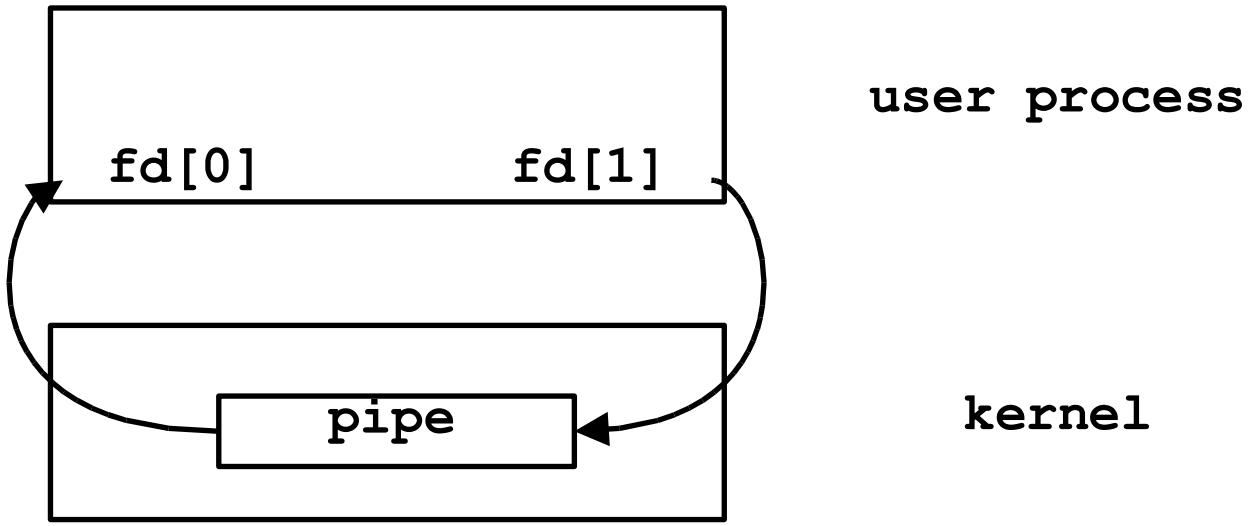
Massimo Verola
(Informatica)

Definizione e caratteristiche di un pipe

- **Cos'è un pipe?**
  - E' un canale di comunicazione che unisce due processi
- **Caratteristiche:**
  - La più vecchia e la più usata forma di *interprocess communication* utilizzata in Unix
  - Limitazioni
    - Sono half-duplex (comunicazione in un solo senso)
    - Utilizzabili solo tra processi con un "antenato" in comune
  - Come superare queste limitazioni?
    - Gli *stream pipe* sono full-duplex
    - *FIFO* (*named pipe*) possono essere utilizzati tra più processi
    - *named stream pipe* = stream pipe + FIFO

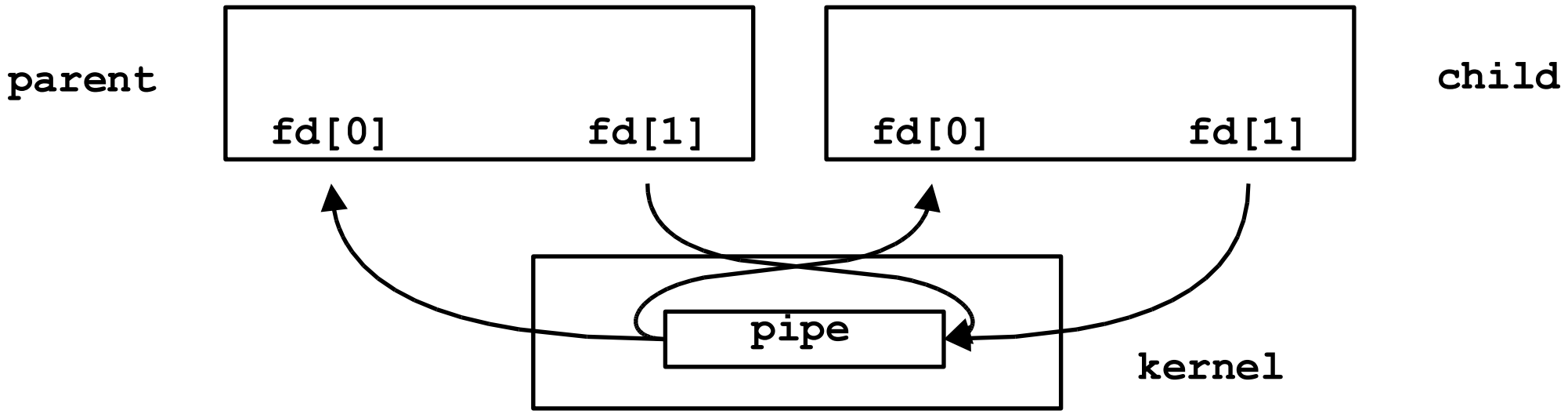
System call `pipe` e file descriptor

- **System call: `int pipe(int filedes[2]);`**
  - Ritorna due descrittori di file attraverso l'argomento **`filedes`**
    - **`filedes[0]`** è aperto in lettura
    - **`filedes[1]`** è aperto in scrittura
  - L'output di **`filedes[1]`** (estremo di write del pipe) è l'input di **`filedes[0]`** (estremo di read del pipe)



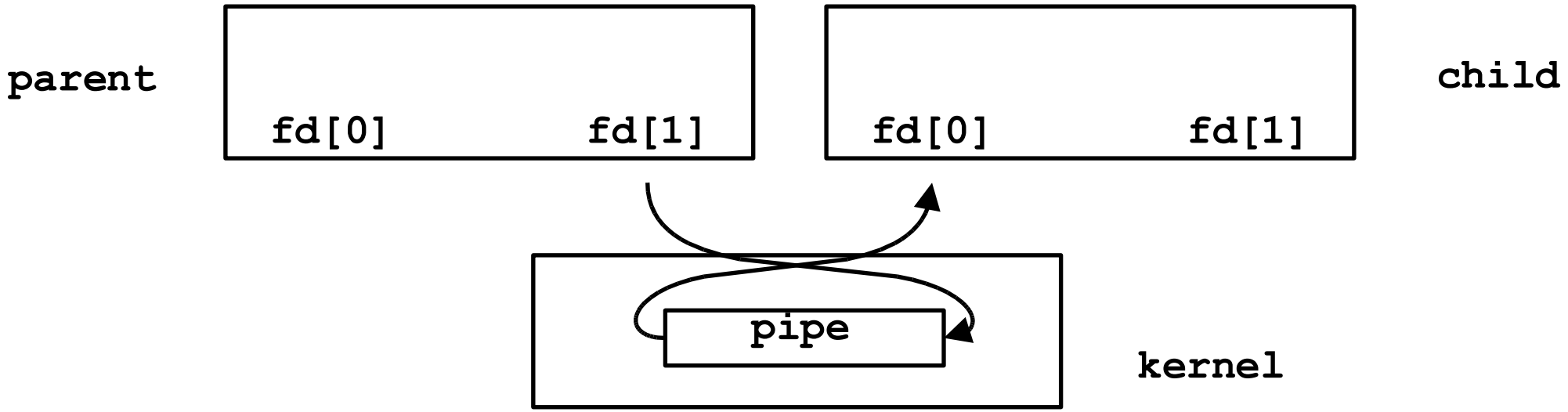
Utilizzo di pipe - I

- **Come utilizzare i pipe?**
  - I pipe in un singolo processo sono completamente inutili
  - Normalmente:
    - il processo che chiama **`pipe`** chiama **`fork`**
    - i descrittori vengono duplicati e creano un canale di comunicazione, **allocato nel kernel**, tra parent e child o viceversa



Utilizzo di pipe - II

- **Come utilizzare i pipe?**
  - Cosa succede dopo la **`fork`** dipende dalla direzione dei dati
  - I canali non utilizzati vanno chiusi
- **Esempio: parent → child**
  - Il parent chiude l'estremo di read (**`close(fd[0]);`**)
  - Il child chiude l'estremo di write (**`close(fd[1]);`**)



Utilizzo di pipe - III

- **Come utilizzare i pipe?**
  - Una volta creati, è possibile utilizzare le normali chiamate **`read/write`** sugli estremi
- **La chiamata `read`**
  - se l'estremo di write è aperto
    - restituisce i dati disponibili, ritornando il numero di byte
    - successive chiamate si bloccano fino a quando nuovi dati non saranno disponibili
  - se l'estremo di write è stato chiuso
    - restituisce i dati disponibili, ritornando il numero di byte
    - successive chiamate ritornano 0, per indicare la fine del file

Utilizzo di pipe - IV

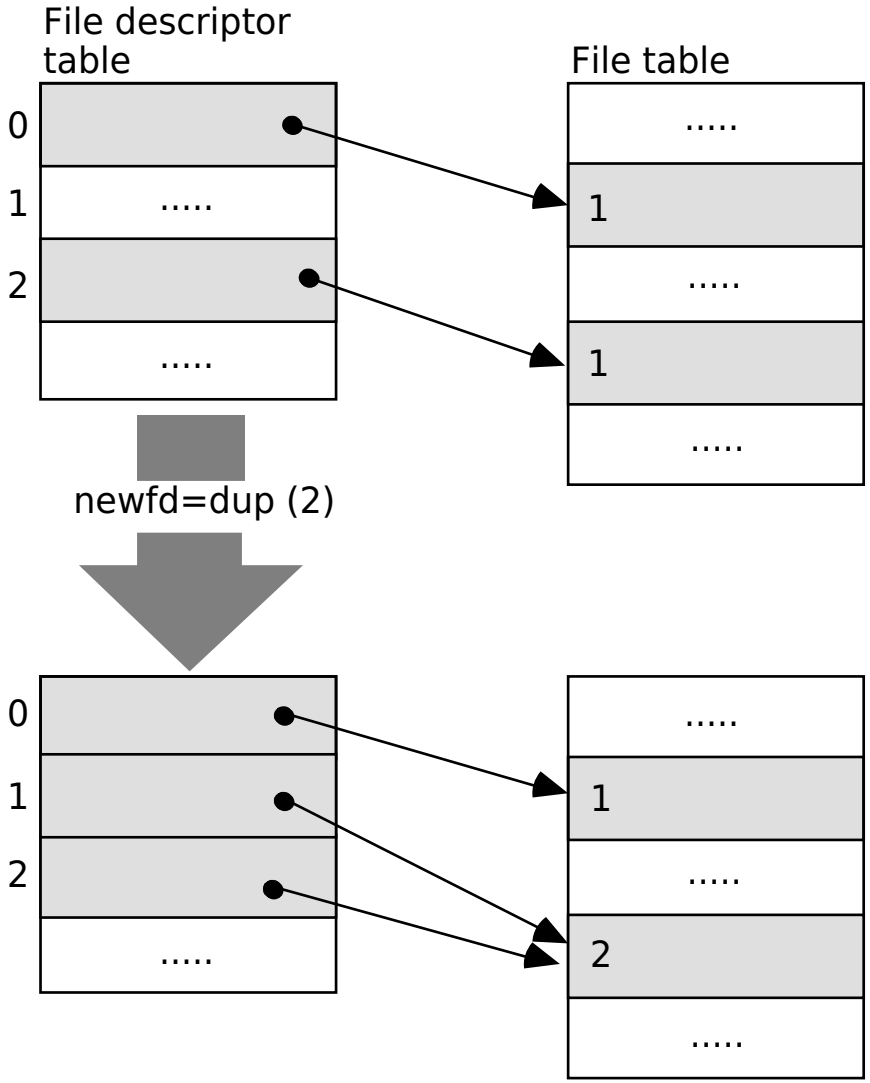
- **La chiamata `write`**
  - se l'estremo di read è aperto
    - i dati in scrittura vengono bufferizzati fino a quando non saranno letti dall'altro processo
  - se l'estremo di read è stato chiuso
    - viene generato un segnale **`SIGPIPE`**
      - ignorato/catturato: write ritorna **`-1`** e **`errno=EPIPE`**
      - azione di default: terminazione
- **Esercizio:**
  - Due processi: parent e child
  - Il processo parent comunica al figlio una stringa, e questi provvede a stamparla

Utilizzo di pipe - V

- **Chiamata `fstat`**
  - Se utilizziamo **`fstat`** su un descrittore aperto su un pipe, il tipo del file sarà descritto come fifo (macro **`S_ISFIFO`**)
- **Atomicità**
  - Quando si scrive su un pipe, la costante **`PIPE_BUF`** specifica la dimensione del buffer del pipe
  - Chiamate **`write`** di dimensione inferiore a **`PIPE_BUF`** vengono eseguite in modo atomico
  - Chiamate **`write`** di dimensione superiore a **`PIPE_BUF`** possono essere eseguite in modo non atomico
    - La presenza di scrittori multipli può causare interleaving tra chiamate **`write`** distinte

Copia del file descriptor - I

- Un file descriptor esistente viene duplicato da una delle seguenti funzioni:
  - `int dup(int filedes);`
  - `int dup2(int filedes, int filedes2);`
- Entrambe le funzioni “duplicano” un file descriptor, ovvero creano un nuovo file descriptor che punta alla stessa file table entry del file descriptor originario
- Nella file table entry c’e’ un campo che registra il numero di file descriptor che la “puntano”



Copia del file descriptor - II

- Funzione `dup`
  - Seleziona il più basso file descriptor libero della tabella dei file descriptor
  - Assegna la nuova file descriptor entry al file descriptor selezionato
  - Ritorna il file descriptor selezionato
- Funzione `dup2`
  - Con **`dup2`**, specifichiamo il valore del nuovo descrittore come argomento **`filedes2`**
  - Se **`filedes2`** è già aperto, viene chiuso e sostituito con il descrittore duplicato
  - Ritorna il file descriptor selezionato

Utilizzo congiunto di `pipe` e `dup`

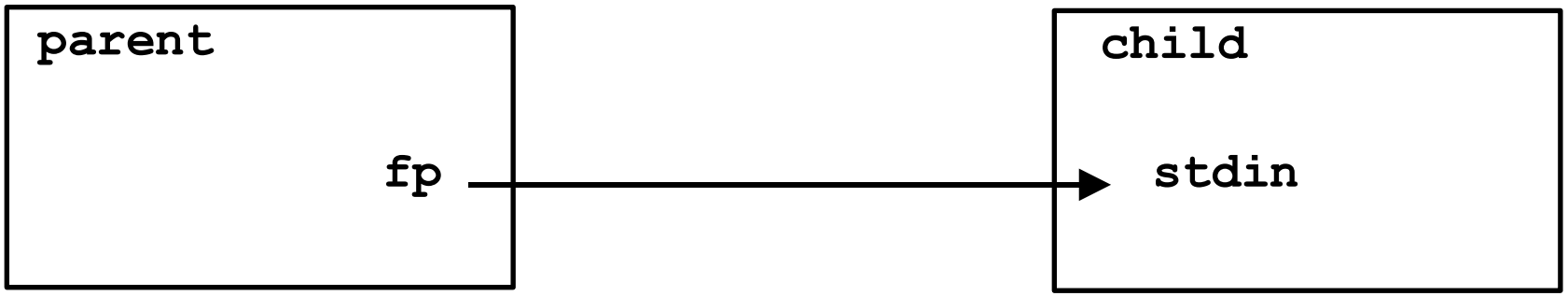
- **Problema**: Consideriamo un programma **`prog1`** che scrive su standard output. Come si puo' fare in modo che l'output venga visualizzato una pagina alla volta, senza pero' modificare il programma stesso?
- **Soluzione:** si scrive un altro programma che:
  - crea un pipe e poi genera un processo child mediante `fork`
  - nel codice del parent chiude l'estremo di read del pipe e lo stdout, e riassegna mediante **`dup2`** il fd dello **`stdout (1)`** sull'estremo di write del pipe
  - nel codice del child chiude l'estremo di write del pipe e lo stdin, e riassegna mediante **`dup2`** il fd dello **`stdin (0)`** sull'estremo di read del pipe
  - il parent mediante **`exec`** lancia il programma **`prog1`**
  - il child mediante **`exec`** lancia un programma tipo di paginazione dell'output tipo **`more`** o **`less`**

popen - I

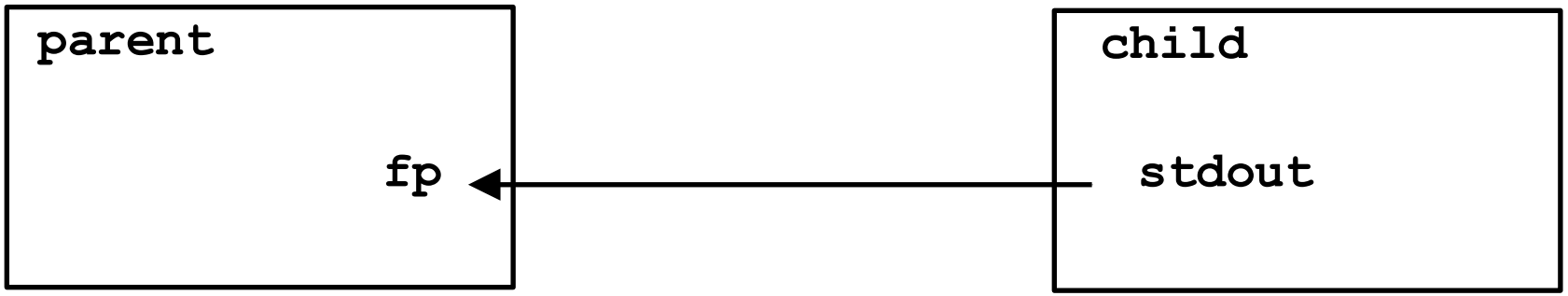
- `FILE *popen(char *cmdstring, char *type);`
- **Descrizione di `popen`:**
  - crea un pipe
  - crea mediante `fork` un processo child
  - chiude gli estremi non utilizzati del pipe
  - esegue mediante `exec` una shell (`sh -c` *`cmdstring`*) per eseguire il comando **`cmdstring`**
  - ritorna uno standard I/O file pointer:
    - se si specifica **`type="r"`** il file pointer (usato in lettura) e' collegato allo standard output del processo child `cmdstring`
    - se si specifica **`type="w"`** il file pointer (usato in scrittura) e' collegato allo standard output del processo child `cmdstring`

popen - II

- `type = "w"`



- `type = "r"`



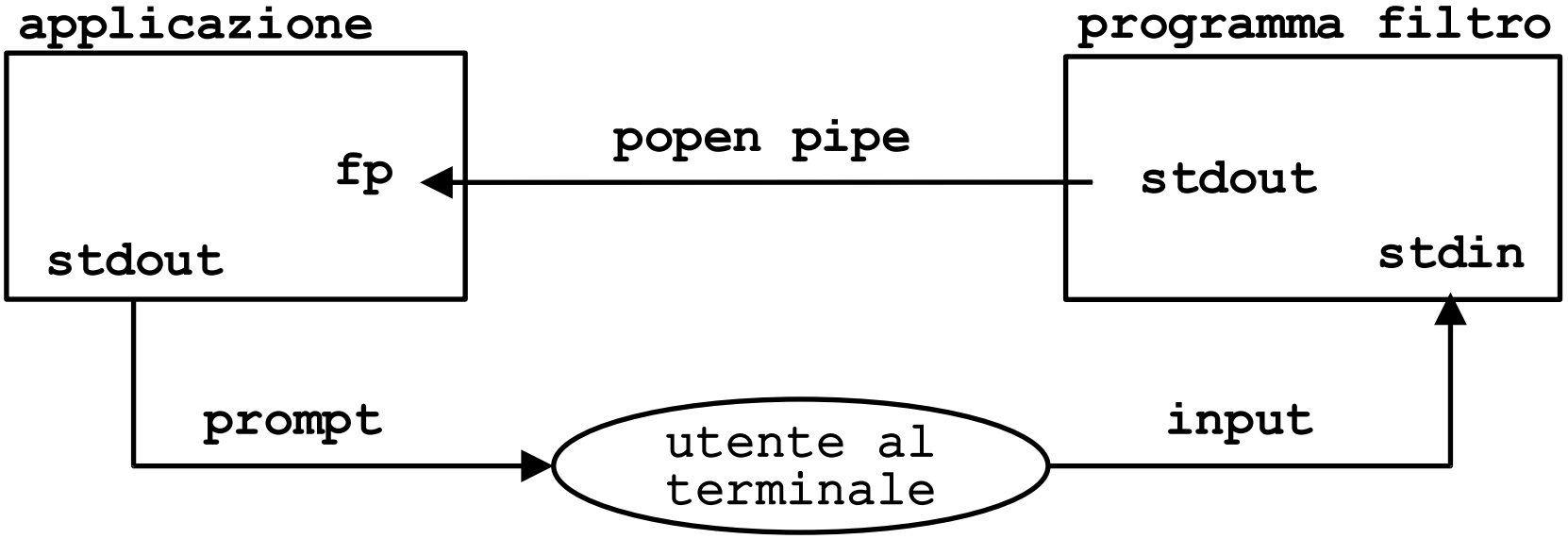
Nota: `cmdstring` è eseguita tramite `"/bin/sh -c"`

pclose - esempio

- `int pclose(FILE *fp);`
- **Descrizione di `pclose`**
  - chiude lo standard I/O file pointer ritornato da `popen`
  - attende mediante **`wait`** la terminazione del comando
  - ritorna il termination status della shell invocata per eseguire il comando

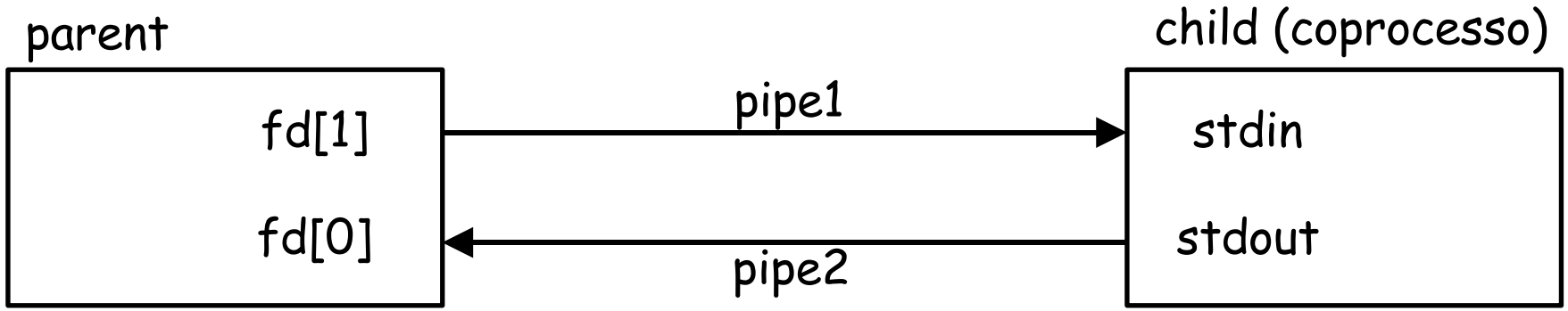
popen - esempio di utilizzo

- Si consideri un'applicazione che scrive un prompt su standard output e legge una linea da standard input
- Mediante popen e' possibile inserire programma ("filtro") tra l'input e l'applicazione, cosi' da trasformare l'input prima che venga letto dall'applicazione
- La trasformazione potrebbe essere l'implementazione della *pathname expansion* o del meccanismo di *history*



Coprocessi

- **Cos'è un coprocesso?**
  - Un **filtro** UNIX è un processo che legge da stdin e scrive su stdout
  - Normalmente i filtri UNIX sono connessi linearmente mediante la pipeline della shell
  - Un filtro si definisce **coprocesso** quando e' collegato ad un altro processo, il quale genera l'input del coprocesso (stdin) e legge l'output del coprocesso (stdout)



Pipe e named pipe

- **Pipe "normali"**
  - possono essere utilizzate solo da processi che hanno un "antenato" in comune, poiche' questo e' l'unico modo per ereditare descrittori di file
- **Named pipe o FIFO**
  - permettono a processi non collegati di comunicare
  - sebbene siano dei canali di comunicazione **allocati nel kernel** come le pipe normali, utilizzano il file system per "dare un nome" ai pipe (**i dati NON vengono scritti su disco!**)
  - un FIFO e' un tipo di file speciale, infatti utilizzando le chiamate `stat, lstat` sul pathname che corrisponde ad un FIFO, la macro `S_ISFIFO` restituirà `true`
  - la procedura per creare un FIFO è simile alla procedura per creare un file

FIFO - I

```
int mkfifo(char* pathname, mode_t mode);
```

- crea un FIFO dal **pathname** specificato
- la specifica dell'argomento **mode** è identica a quella di **open**, **creat** (mode codifica i permessi di accesso al file mediante un numero ottale, ad esempio `0644 = rw-r--r--`)

- **Come funziona un FIFO?**
  - una volta creato un FIFO, le normali chiamate **open**, **read**, **write**, **close**, possono essere utilizzate per leggere il FIFO
  - il FIFO può essere rimosso utilizzando **unlink**
  - le regole per i diritti di accesso si applicano come se fosse un file normale

Leggi: `man 4 fifo` per ulteriori informazioni e descrizione del comportamento specifico delle varie system call

FIFO - II

- **Chiamata open**
  - File aperto senza flag O_NONBLOCK
    - Se il FIFO è aperto in sola lettura, la chiamata **si blocca** fino a quando un altro processo non apre il FIFO in scrittura
    - Se il FIFO è aperto in sola scrittura, la chiamata **si blocca** fino a quando un altro processo non apre il FIFO in lettura
  - File aperto con flag O_NONBLOCK
    - Se il FIFO è aperto in sola lettura, la chiamata **ritorna immediatamente**
    - Se il FIFO è aperto in sola scrittura, e nessun altro processo lo ha aperto in lettura, la chiamata **ritorna un messaggio di errore**

FIFO - III

- **Chiamata `write`**
  - se nessun processo ha aperto il file in lettura viene generato un segnale **`SIGPIPE`**:
    - ignorato/catturato: write ritorna **`-1`** e **`errno=EPIPE`**
    - azione di default: terminazione
- **Atomicità**
  - Quando si scrive su un pipe, la costante **`PIPE_BUF`** (in genere pari a 4096, vedi `/usr/include/linux/limits.h`) specifica la dimensione del buffer del pipe
  - Chiamate **`write`** di dimensione inferiore a **`PIPE_BUF`** vengono eseguite in modo atomico
  - Chiamate **`write`** di dimensione superiore a **`PIPE_BUF`** possono essere eseguite in modo non atomico
  - La presenza di piu' scrittori può causare interleaving tra chiamate **`write`** distinte

FIFO - IV

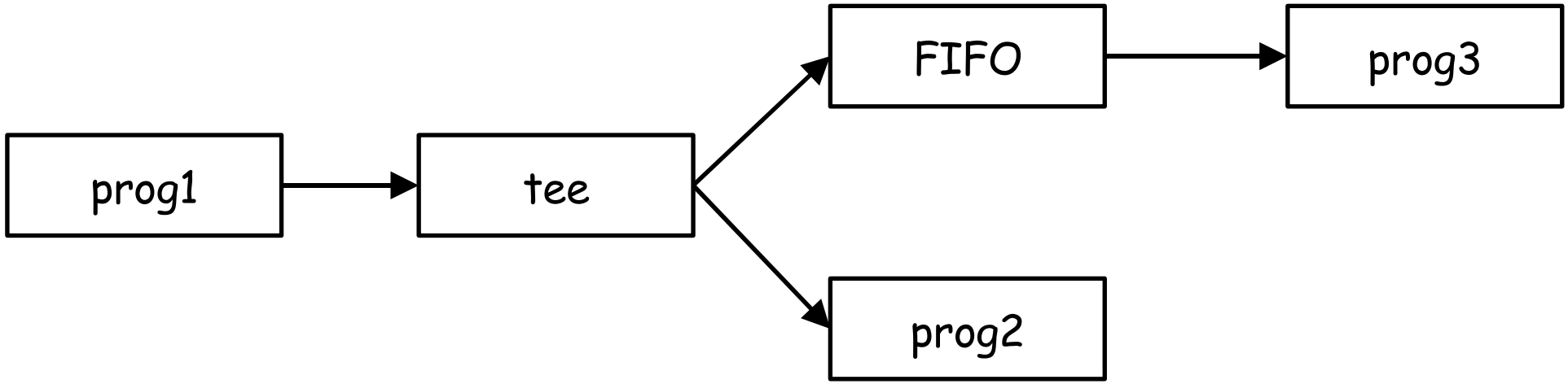
Tabella riassuntiva sull'effetto del flag `O_NONBLOCK` su pipe e FIFO

CONDIZIONE	COMPORTAMENTO DI DEFAULT	COMPORTAMENTO CON O_NONBLOCK
open di FIFO read-only senza che altri processi abbiano il FIFO aperto in scrittura	attesa finche' un processo apre FIFO in scrittura	ritorno immediato senza errore
open di FIFO write-only senza che altri processi abbiano il FIFO aperto in lettura	attesa finche' un processo apre FIFO in lettura	ritorno immediato con errore, errno pari a ENXIO
read da pipe o FIFO che non contiene dati	attesa finche' non vi siano dati in FIFO, o finche' nessun processo abbia piu' FIFO aperto in scrittura	ritorno immediato, valore di ritorno pari a 0
write in pipe o FIFO pieni	attesa finche' non vi sia spazio per scrivere, dopodiche' scrittura dei dati	ritorno immediato, valore di ritorno pari a 0

FIFO - V

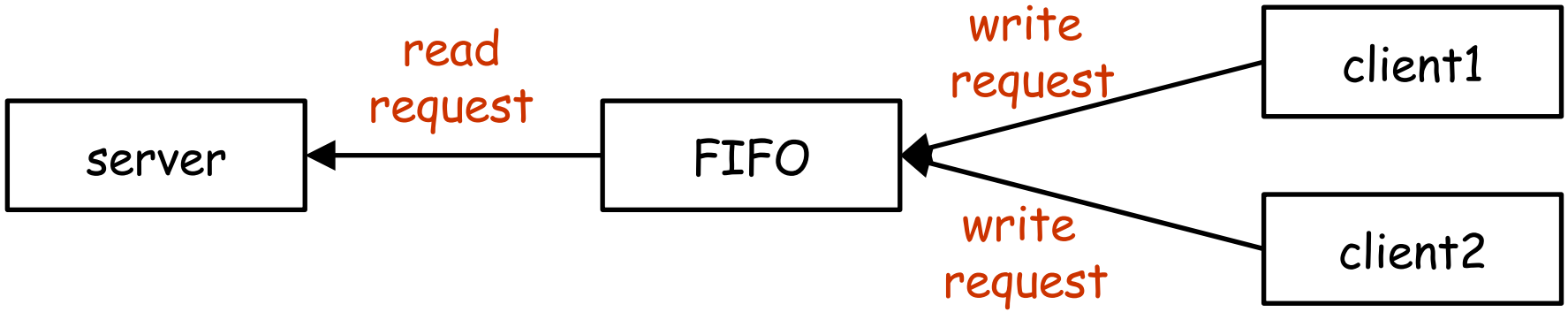
- **Utilizzo dei FIFO**
  - Utilizzati dai comandi shell per passare dati da una shell pipeline ad un'altra, senza passare creare file intermedi
- **Esempio:**

```
mkfifo fifo1
prog3 < fifo1 &
prog1 | tee fifo1 | prog2
```



FIFO - VI

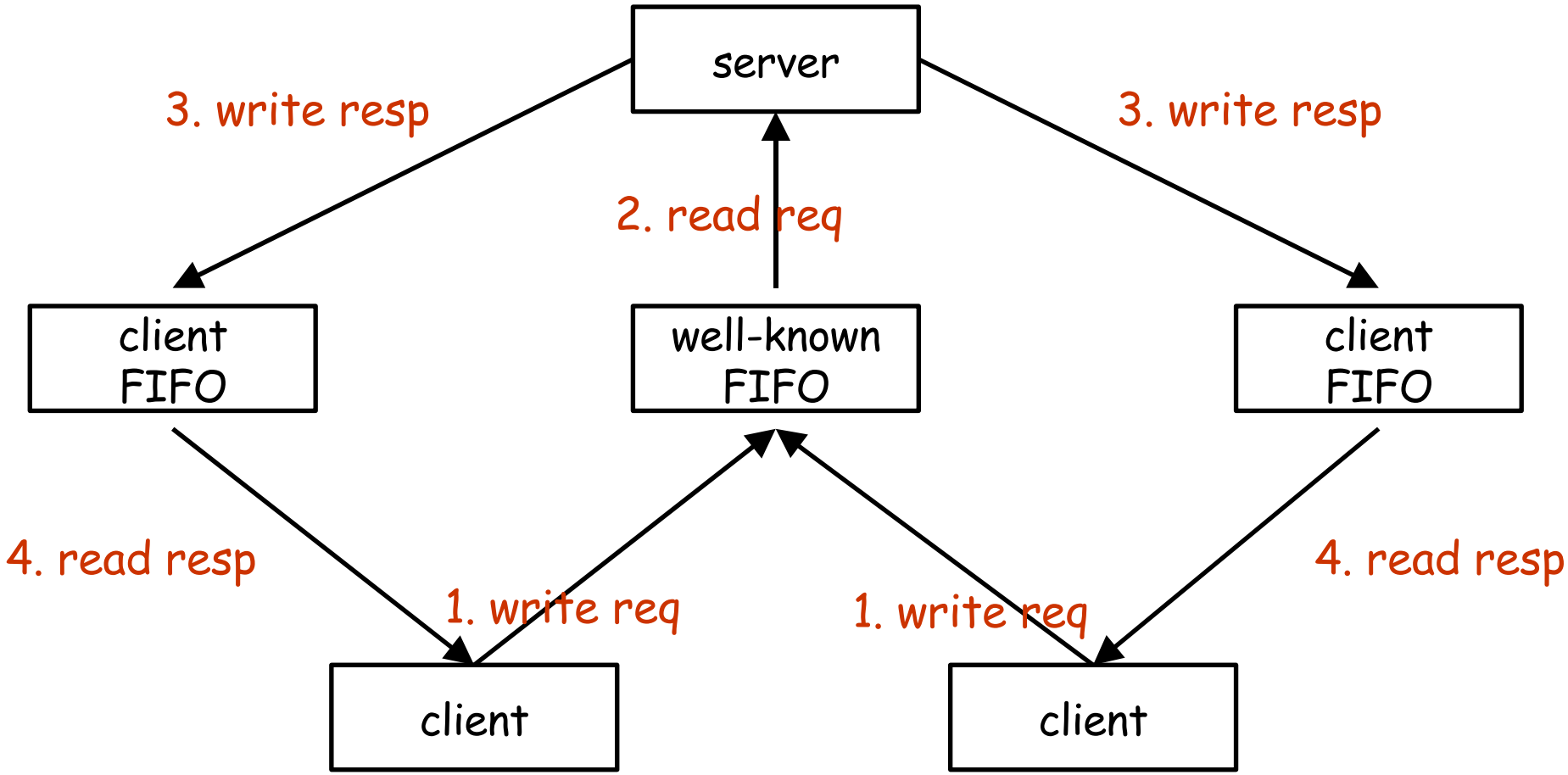
- **Utilizzo dei FIFO**
  - Utilizzati nelle applicazioni client-server per comunicare
- **Esempio:**
  - Comunicazioni client → server
    - il server crea un FIFO
    - il pathname di questo FIFO deve essere "well-known" (ovvero, noto a tutti i client)
    - i client scrivono le proprie richieste sul FIFO
    - il server leggere le richieste dal FIFO



FIFO - VII

- **Problema: come rispondere ai client?**
  - Non è possibile utilizzare il "well-known" FIFO
    - I client non saprebbero quando leggere le proprie risposte
  - Soluzione:
    - i client spediscono il proprio process id al server
    - ogni client crea un proprio FIFO (*client FIFO*) per la risposta, il cui nome contiene il process ID (in modo tale che il server puo' ricostruirlo), e lo apre in lettura
    - il server apre in scrittura il *client FIFO*
    - il server scrive sul *client FIFO* la risposta alla richiesta del client
  - Suggerimenti:
    - Il server dovrebbe catturare SIGPIPE, in quanto il client potrebbe terminare o chiudere il FIFO prima di leggere la risposta (altrimenti il SIGPIPE provocherebbe la terminazione del server)
    - Il server dovrebbe aprire in lettura e scrittura il "well-known" FIFO, altrimenti, quando l'ultimo client termina, il server leggerà EOF, invece di rimanere bloccato sulla read, in attesa che un nuovo client si connetta sulla "well-known FIFO"

FIFO - VIII



Esercitazioni

- **Esercizio 1:**
  scrivere un programma **`test_fifo.c`** per verificare i quattro casi possibili di configurazione di una FIFO: read oppure write, con o senza il flag O_NONBLOCK.
- **Esercizio 2:**
  scrivere un programma che esemplifica l'interazione <Produttore-Consumatore>:
  - Utilizzare la named pipe (FIFO) come buffer
  - Il produttore scrive interi sulla pipe e il consumatore li stampa
  - Utilizzare anche più produttori
- **Esercizio 3:**
  scrivere un programma che estende lo schema di comunicazione della pagina precedente, creando un server dedicato (mediante **`fork`**) per ogni client e una FIFO tra client (in write) e server dedicato (in read). Il client scrive sulla FIFO la linea inserita da stdin e il server dedicato la legge da FIFO e la stampa su stdout.