Laboratorio di Sistemi Operativi

Marzo-Giugno 2008
Matricole congrue 0 mod 3

IPC: pipe e FIFO (1)

Forme di comunicazione tra processi

La comunicazione tra processi può avvenire:

- Passando dei files aperti tramite fork
- Attraverso il filesystem
- Utilizzando le **pipe**
- Utilizzando le **FIFO**
- Utilizzando **IPC di System V** (code di messaggi, semafori e memoria condivisa)
- Utilizzando **stream** e **socket** (per processi su hosts differenti)

Caratteristiche delle pipe

- Le pipe sono half-duplex
  - flusso di dati in una sola direzione
- Possono essere utilizzate solo tra processi che hanno un antenato in comune
  - di solito la pipe è creata da un processo che successivamente chiama una fork; la pipe viene usata per le comunicazioni tra padre e figlio

Funzione pipe

```
#include <unistd.h>

int pipe(int *filedes);
```

Restituisce: 0 se OK
-1 in caso di errore

Attraverso l'argomento *filedes* restituisce 2 file descriptor

pipe

- in *filedes[0]* c'è un file descriptor di un "file" aperto in lettura
- in *filedes[1]* c'è un file descriptor di un "file" aperto in scrittura

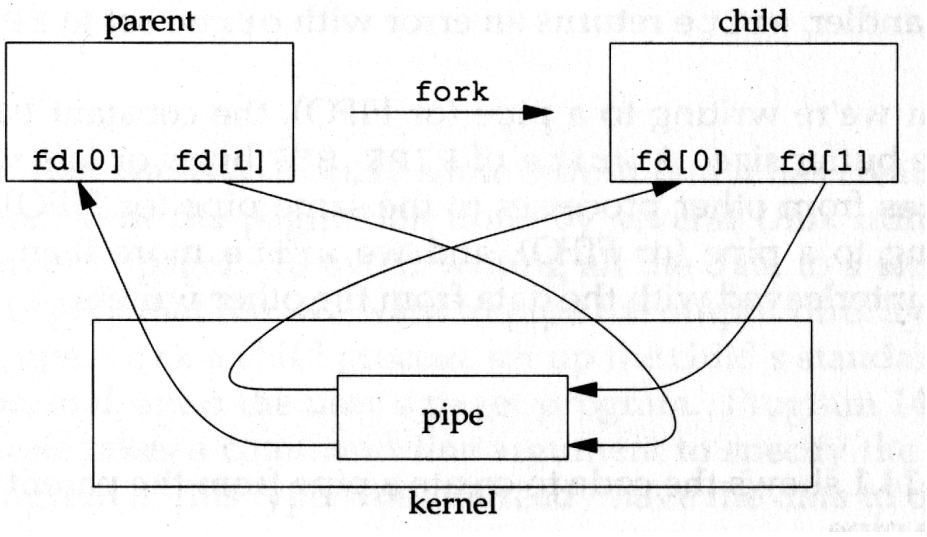
- I dati fluiscono dal file in cui si scrive a quello in cui si legge andando attraverso il kernel grazie alla pipe
  - l'output di *filedes[1]* è l'input di *filedes[0]*

Utilizzo delle pipe (1)

- L'utilizzo tipico delle pipe è il seguente

```
int fd[2];
…
pipe(fd);
pid=fork();
…
```

Situazione dopo pipe+fork

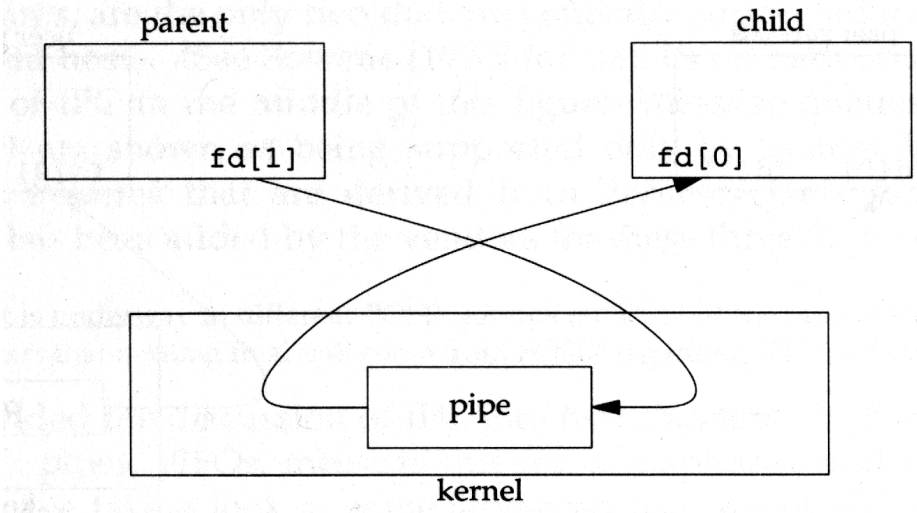


Utilizzo delle pipe (2): scelta della direzione

- Una delle possibilità dopo la fork è di creare un canale dal padre verso il figlio:

```
if(pid>0) {          /* padre */
   close(fd[0]);
}else{                  /* figlio */
   close(fd[1]);
}
```

Pipe da padre a figlio



Utilizzo delle pipe (3)

- Una volta che è stata creata la pipe e che è stato scelto il verso di comunicazione è possibile utilizzare le funzioni di I/O che lavorano con i file descriptor (e.g. ***read***, ***write***)
- Una pipe è un canale di comunicazione in cui i dati vengono letti nello stesso ordine in cui vengono scritti

I/O su una pipe

- Funzione **write**
  - Quando la pipe si riempie (la costante PIPE_BUF specifica la dimensione) la **write** si blocca fino a che la **read** non ha rimosso un numero sufficiente di dati (non vengono effettuate scritture parziali).
  - Se il descrittore del file che legge dalla pipe è chiuso, una **write** genererà un errore (segnale SIGPIPE)

I/O su una pipe

- Funzione **read**
  - Legge i dati dalla pipe nell'ordine in cui sono scritti. Non è possibile rileggere o rimandare indietro i dati letti.
  - Se la pipe è vuota la **read** si blocca fino a che non vi siano dei dati disponibili.
  - Se il descrittore del file in scrittura è chiuso, la **read** restituirà EOF dopo aver completato la lettura dei dati.

I/O su una pipe

- Funzione **close**
  - La funzione **close** sul descrittore del file in scrittura agisce come *end-of-file* per la **read**.
  - La chiusura del descrittore del file in lettura causa un errore nella **write**.


```
int fd[2];

pipe(fd);
pid=fork();
if(pid>0) {          /* padre */
   close(fd[0]);
   write(fd[1], "ciao figlio\n",12);
}else{                  /* figlio */
   close(fd[1]);
   n=read(fd[0], line, 12);
   write(STDOUT_FILENO, line, n);
}
```

Nota

- una cosa interessante è duplicare i descrittori della pipe su *standard input* ed *output*
- a questo punto il figlio esegue con una *exec* un programma che può leggere da standard I/O

Esempio: simulazione della pipe | sotto bash

Esempio: `$ cat file | more`

Come vengono eseguiti i due processi relativi all'esecuzione dei due comandi?

Il comando **cat** ha come input il file e come output STDOUT.

Il comando **more** ha come input un file e come output STDOUT.

Come vengono modificati STDIN e STDOUT di questi comandi?

Nel processo **shell** viene creata la **pipe**

pipe(fd);

0	STDIN
1	STDOUT
2	STDERR
3	PIPE_R
4	PIPE_W
5	
6	……..

Si eseguano due **fork** (la prima dovrà eseguire cat file, la seconda more)

Nel primo figlio viene chiuso il descrittore in lettura

close(fd[0]);

0	STDIN
1	STDOUT
2	STDERR
3	PIPE_R
4	PIPE_W
5	
6	……..

Successivamente viene chiuso lo standard output

close(1);

0	STDIN
1	STDOUT
2	STDERR
3	
4	PIPE_W
5	
6	……..

Con la funzione **dup** lo standard output coinciderà con il descrittore della pipe in scrittura

dup(fd[1]);

0	STDIN
1	PIPE_W
2	STDERR
3	
4	PIPE_W
5	
6	……..

Si chiude il descrittore del file in scrittura

close(fd[1]);

0	STDIN
1	PIPE_W
2	STDERR
3	
4	PIPE_W
5	
6	……..

Si esegue la **exec** del comando **cat file**

exec(cat file);

0	STDIN
1	PIPE_W
2	STDERR
3	
4	
5	
6	……..

Nel secondo figlio avremo la stessa tabella dei file descriptor

0	STDIN
1	STDOUT
2	STDERR
3	PIPE_R
4	PIPE_W
5	
6	……..

Viene chiuso il descrittore del file in scrittura

close(fd[1]);

0	STDIN
1	STDOUT
2	STDERR
3	PIPE_R
4	PIPE_W
5	
6	……..

Viene chiuso lo standard input

close(0);

0	STDIN
1	STDOUT
2	STDERR
3	PIPE_R
4	
5	
6	……..

Con la **dup** lo standard input coinciderà con il descrittore del file in lettura

dup(fd[0]);

0	PIPE_R
1	STDOUT
2	STDERR
3	PIPE_R
4	
5	
6	……..

Viene chiuso il descrittore del file in lettura

close(fd[0]);

0	PIPE_R
1	STDOUT
2	STDERR
3	PIPE_R
4	
5	
6	……..

Viene eseguita la **exec** del comando **more**

exec(more);

0	PIPE_R
1	STDOUT
2	STDERR
3	
4	
5	
6	……..

Esercizio

Scrivere un programma che:

- prende in input il nome di un file di testo
- crea due figli che comunicano tramite pipe
- il primo figlio esegue ***cat file*** e manda l'output al fratello tramite la pipe
- il secondo figlio visualizza a video le informazioni ricevute dalla pipe con il comando ***more***

Esercizio

- Ripetere l'esercizio di copia di un file attraverso 2 processi che copiano vocali e consonanti, rispettivamente, implementando la sincronizzazione attraverso le pipe. Si implementino le funzioni TELL e WAIT che usano le pipe.

Esercizio

Un processo P1 crea una pipe e un figlio F1.

Un secondo processo P2 comunicherà con P1 tramite un file TEMP.

P2 ogni secondo genera un numero casuale da 1 a 128 e lo scrive in TEMP seguito dal proprio pid.

P1 dopo 20 secondi dalla creazione del figlio scrive nella pipe il pid di P2, seguito dal numero -1, poi stampa un messaggio sullo schermo e termina la sua esecuzione. Durante questi 20 secondi P1 leggerà i numeri nel file TEMP e scriverà sulla pipe il suo pid con il numero che ha letto.

F1 leggerà dalla pipe i pid seguiti dal numero. Se il numero è -1 ucciderà P2 e poi terminerà; altrimenti stamperà al terminale il proprio pid seguito dal numero che ha letto.