Lezione 12: PIPE

Lezione 12: Pipe

12.1

La gestione della sincro in unix riguarda due apsetti; scambio di dati e segnalazione di eventi-

Lo scambio avviene attraverso

- **pipe →** operazioni di i o su code FIFO sincronizzate dal SO.
- **messaggi →** invio e ricezione di messagi tipizzati su coda FIFO.
- **memoria condivisa →** allocata e associata al processo attraverso SC.

La segnalazione di azioni avviene attraverso segnali e semafori.

12.2 IPC

Meccanisci che permettonio a processi distinti di comunicare e scambiare info. I peocessi che comunicano possono risiedere sulla stessa macchina o su macchine diverse. LA comunicazione può essere finalizzata a **cooperazione** ovvero i processi scambiano dati per l'ottenimento di un fine comune o **sincro** ovvero lo scambio avviene

12.3 Pipe anonime

Questo termine è usato per indicare situazioni in cui si connette un flusso di dati da un processo ad un altro

I due processi connessi da una pipe sono eseguiti concorrentemente, memorizza automaticamente l'out dello scrittore cmd1 in un buffer. Se il buffer è pieno lo scrittore si sospendèfino a che alcuni dati non venfono letti. Se il buffer è vuoto il lettore cmd2 si sospende fino a che diventano disponibili dei dati in out.

```
cmd1 | cmd2
```

Sono presenti in tutte le versioni di unix, utilizzate ad esempio dalle shell per le pipeline.

Pipe con nome o FIFO sono prensenti in unix system v in poi

Una pipe anonima è un canale di comunicazione creato con `pipe()` mantenuto a livello kernel che unisce due processi. è unidirezionale e permette la comunicazione solo tra processi con un antenato in comune.

Presenta inoltre due lati di accesso, ciascuno associato ad un descrittore di file. Il lato di lettura è acceduto invocando `read()` il lato di scrittura è acceduto invocando `write()`, memorizzail suo in in un buffer. Quando un processo ha finito di usare u lato di una pipe chiude il descrittore con `close()`.

```
#include <unistd.h>
int pipe (int fd[2])
```

Crea una pipe anonima e restituisce due descrittori di file, lato lettura `fd[0]` e scrittura `fd[1]`

12.3.1 pipe in lettura

Il lato di scrittura è stato chiuso, read restituisce 0 che indica la fine dell'in

Se la pipe è vuota e il lato di scritturàè ancora apero, si sospende.

Se il processo tenta di leggere più byte di quelli presenti, restituisce il numero di byte letti.

12.3.2 pipe in scrittura

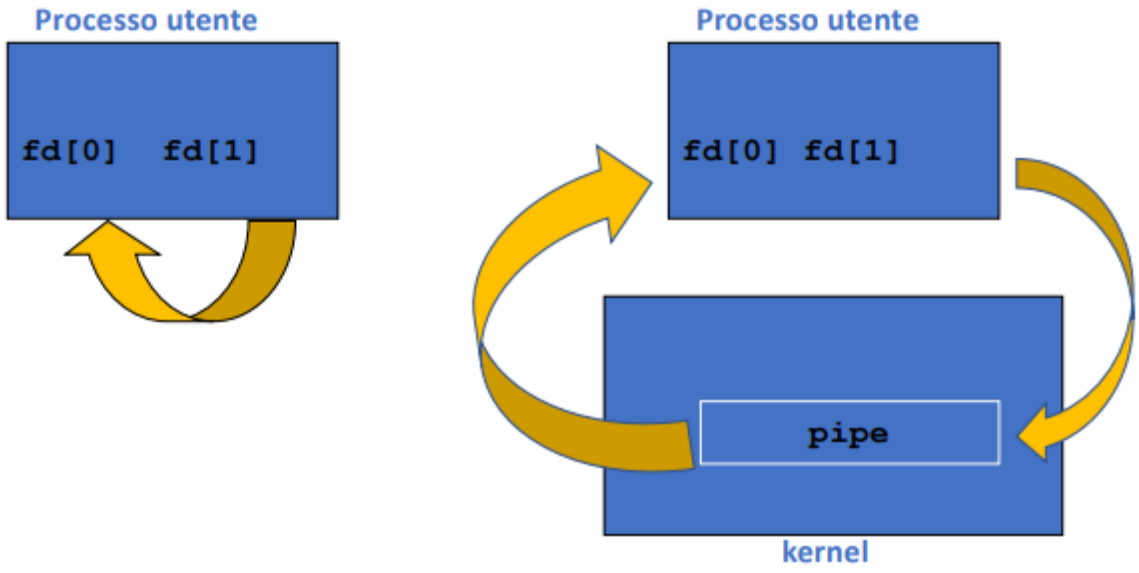
Il lato lettore è stato chiuso, write fallisce ed allo scrittore è inviato un segnale SIGPIPE, la cui azione di default è di far terminare il ricevente.

Se si scrive meno byte di quelli che una pipe può contenere, viene eseguita in modo atomico ovvero non possono avvenire intrecci dei dati scritti da processi diversi.

`lseek()` non ha senso se applicata ad una pipe.

Dato che l'accesso ad una pipe anonuma avviene tramite descrittore di file, solo il processo creatore ed i suoi discendenti possono accedere alla pipe.

12.3.3 Pipe anonime, un singolo processo



sinistra i due estremi di una pipe connessi in un processo.

destra flusso dei dati nella pipe attraverso kernel

la chiamata di sistema `fstat()` restituisce il tipo di fifo peèil descrittorèdi file delle estremintà di una pipe.

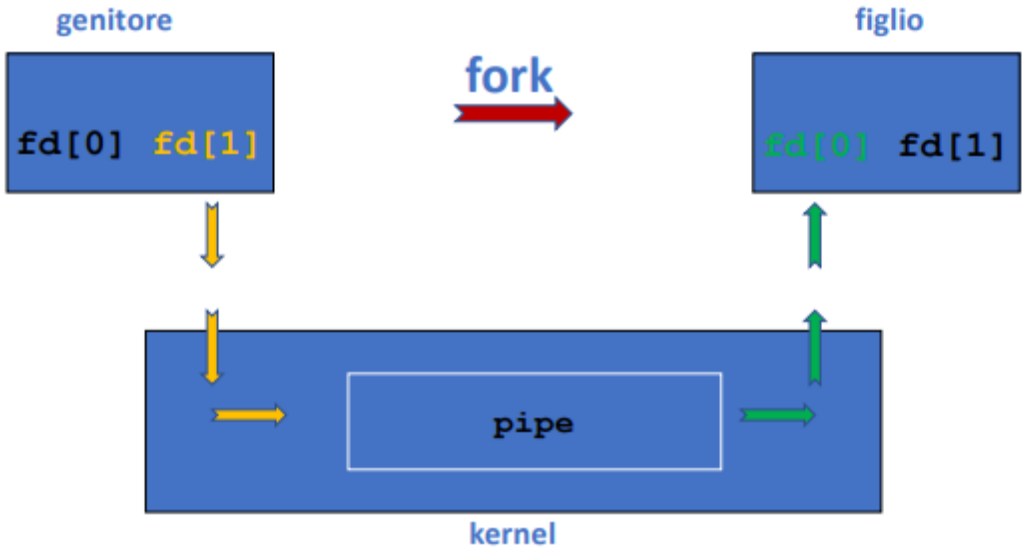
Possiamo verificare il tipo di pipe con la macro S_ISFIFO

utilizzare una pipe in un processo singolo nonha senso, normalmente il processo crea na pipe successivamento invocata una fork()

La tipica sequenza è:

- il processo crea una pipe
- il processo crea un figlio

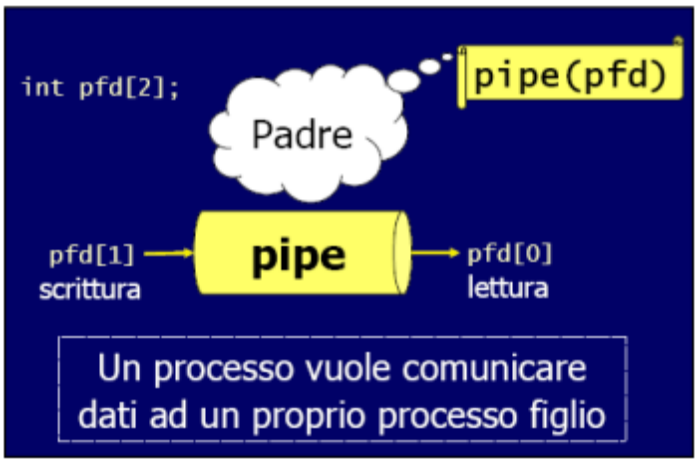
- lo scrittore chiude il lato di lettura e viceversa
- i processi comunicano
- ogni processo chiude il suo descrittore quando ha finito



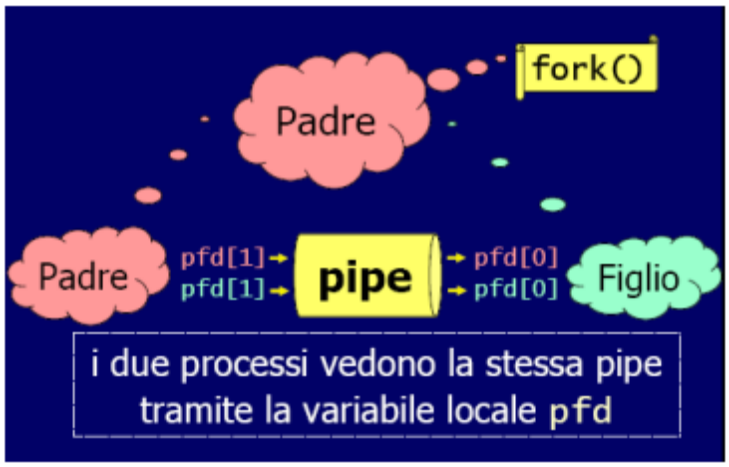
12.4 pipe anonime e fork

Cosa accade dopo la `fork()` dipende dalla direzione del flusso di dati che vogliamo. Per una pipe dal genitore al figlio, il genitore chiude l'estremità in lettura della pipefd[0] ed il figlio chiude l'estremità in scrittura fd[1]. Per una pepe dal figlio al genitore, il genitore chiude fd[1]e il figlio chiude fd[0]

12.5 comunicazione padre figlio



12.6 Esempio: padre scrive - figlio legge



12.7 Esempio: Leggere messaggi dal figlio

```
#include <stdio.h>
#include <string.h>
#define READ 0
#define WRITE 1
```

```
char *frase = "Messaggio...”;
int main() {
    int fd[2], bytesRead;
    char message[100];
    pipe(fd);
    if(fork() == 0) {
    close(fd[READ]);
    write(fd[WRITE],frase,strlen(frase)+1);
    close(fd[WRITE]);
    }
    else {
        close(fd[WRITE]);
        bytesRead=read(fd[READ],message, 100);
        printf("Letti %d byte: %s\n", bytesRead, message);
        close(fd[READ]);
    }
}
```

12.8 Comunicazione tramite pipe

Quando un processo scrittore invia più messaggi di lunghezze variabile tramite una pipe, occorre fissare un protocollo di comunicazione che permetta al processo lettore di individuare la fine di ogni singolo messaggio.

Alcune possibilità sono:

- inviare la lunghezza del messaggio prima del messaggio stesso
- terminare un messaggio con un carattere speciale o un newline

Più in generale il protocollo stabilisce la sequenza di messaggi attesa delle due parti.

12.9 Esempio: Protocol.c

```
/* Semplice protocollo di comunicazione tramite pipe anonima. Do
variabile ogni messaggio è preceduto da un intero che fornisce l

#define READ 0 /* Estremità in lettura della pipe */
#define WRITE 1 /* Estremità in scrittura della pipe */
char *msg[3] = { "Primo", "Secondo", "Terzo" };

int main (void) {
    int fd[2], i, length, bytesRead;
    char buffer [100]; /* Buffer del messaggio */
    pipe (fd); /* Crea una pipe anonima */
    if (fork () == 0) { /* Figlio, scrittore */
        close(fd[READ]); /* Chiude l'estremità inutilizzata */
        for (i = 0; i < 3; i++) {
            length=strlen (msg[i])+1 ; /* include \0 */
            write (fd[WRITE], &length, sizeof(int));
            write (fd[WRITE], msg[i], length);
        }
        close (fd[WRITE]); /* Chiude l'estremità usata */
    }
    else { /* Genitore, lettore */
        close (fd[WRITE]); /* Chiude l'estremità non usata */
        while (read(fd[READ], &length, sizeof(int))) {
            bytesRead = read (fd[READ], buffer, length);
            if (bytesRead != length) {
                printf("Errore!\n");
                exit(1);
            }
            printf("Padre:Letti %d byte:%s\n",bytesRead,buffer)
        }
        close (fd[READ]); /* Chiude l'estremità usata */
    }
```

```
        exit(0);
}
```

12.10 pipe ed exec()

I due processi comunicanti devono conoscere i desccrittori della pipeline. Finora si è sfruttato lo sdoppiamento delle variabili locali ed il fatto che un processo figlio eredita la tabella dei descrittori del padre.

Una exec non altera la tavola dei descrittori anche se s perde lo spazio di memoria ereditato dal padre prima della exec()

supponiamo di voler implementare

`ls | wc - w`

- il padre esegue fork()
- il padre esegue exec() per `ls`
- il figlio esegue exec() per `wc -w`

12.11 dup2 e pipeling

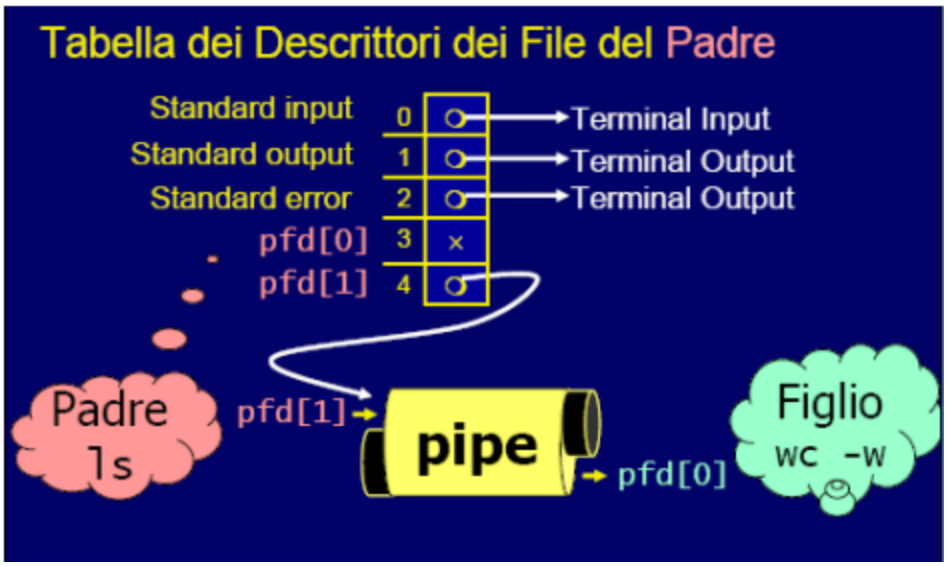
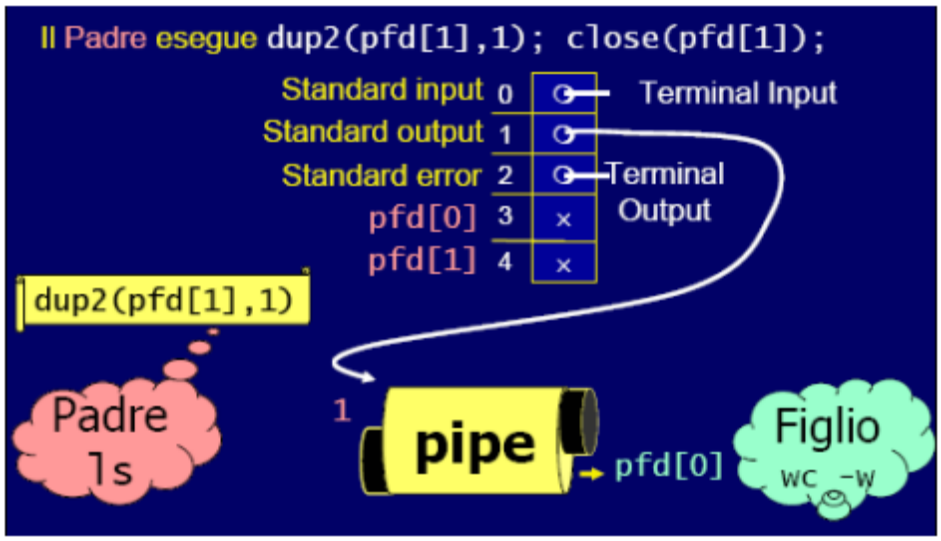
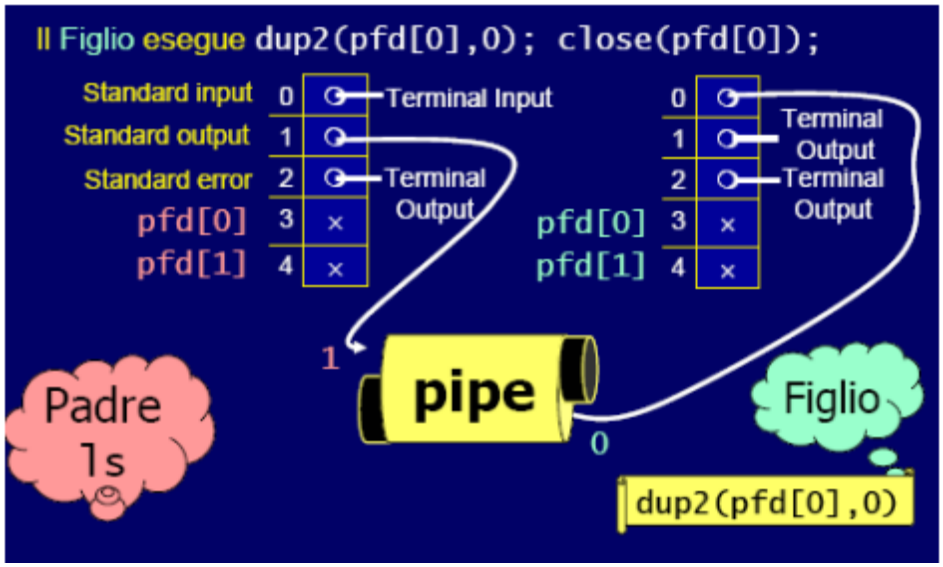
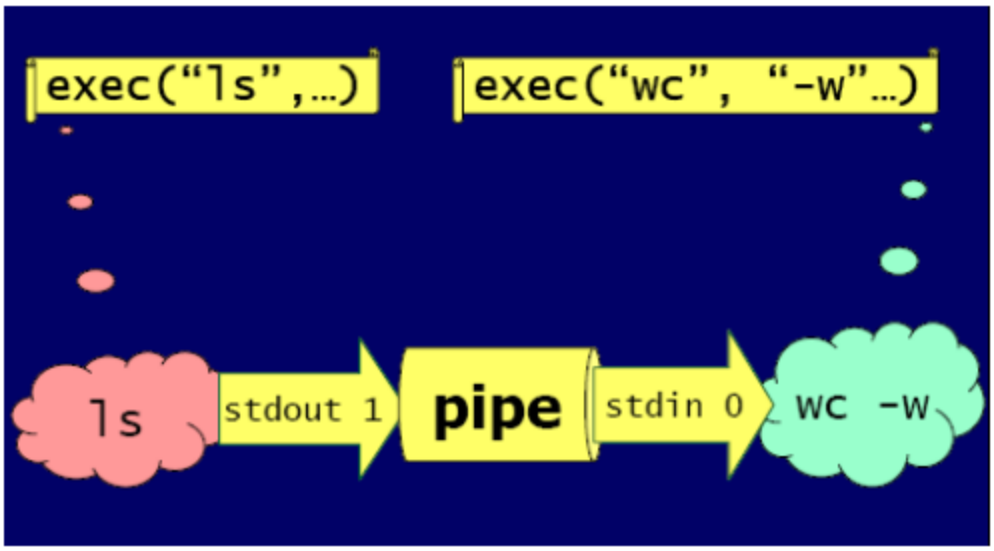


Tabella dei Descrittori dei File del Padre
Standard input
Standard output
Standard error
pfd[0]
pfd[1]
0
1
2
3
4
Terminal Input
Terminal Output
Terminal Output
Padre
ls
pfd[1]
pipe
pfd[0]
Figlio
wc -w


Il Padre esegue dup2(pfd[1],1); close(pfd[1]);
Standard input
Standard output
Standard error
pfd[0]
pfd[1]
0
1
2
3
4
Terminal Input
Terminal Output
dup2(pfd[1],1)
Padre
ls
1
pipe
pfd[0]
Figlio
wc -w





12.12 Esempio 4

Il programma seguente usa dup3 peèinvare out da una pipe al comando sort, dopo aver creato la pipe il programma invoca fork, il processo genitore stampa le stringhe

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
```

```
#include <unistd.h>

int main (){
    int fds[2];
    pid_t pid;

    /* Crea una pipe. I descrittori per le estremità sono in fds
    pipe (fds);

    /* Crea un processo figlio. */
    pid = fork ();
    if (pid == (pid_t) 0) {
        /* Processo figlio. Chiude descrittore lato scrittura */
        close (fds[1]);

        /* Connette il lato lettura della pipe allo standard inp
        dup2 (fds[0], STDIN_FILENO);

        /* Sostituisce il processo figlio con il programma “sort
        execlp (“sort”, “sort”, 0);
    }
    else {/* Processo genitore */
        /* Chiude il descrittore lato lettura */
        close (fds[0]);
        write(fds[1], “This is a test.\n”, 16);
        write(fds[1], “Hello, world.\n”, 14);
        write(fds[1], “My dog has fleas.\n”,18 );
        write(fds[1], “This program is great.\n”,23);
        write(fds[1], “One fish, two fish.\n”,20);
        close (fds[1]);
        /* Aspetta che il processo figlio finisca */
        waitpid (pid, NULL, 0);
    }
return 0;
}
```

12.13 Esempio 5

```
#include <stdio.h>
#include <unistd.h>

/* Esempio di creazione pipe per lanciare ls | sort */
int main(){
    int pid;
    int fd[2];
    pipe(fd);
    if ((pid = fork()) == 0)
        { /* figlio */
            /* chiusura lettura da pipe */
            close(fd[0]);
            /* redirezione stdout a pipe */
            dup2(fd[1], 1);
            execlp("ls", "ls", NULL);
        } else if (pid > 0)
            { /* padre */
                /* chiusura scrittura su pipe */
                close(fd[1]); /* redirezione stdin a pipe */
                dup2(fd[0],0);
                execlp("sort", "sort", NULL);}
}
```

12.13 Esempio redirezione con pipe

```
#include <stdio.h>
#define READ 0
#define WRITE 1

int main (int argc, char *argv []) {
    int fd [2];
    pipe (fd); /* Crea una pipe senza nome */
```

```
    if (fork () != 0) { /* Padre, scrittore */
        close (fd[READ]); /* Chiude l'estremità non usata */
        dup2 (fd[WRITE], 1); /* Duplica l'estremità usata allo s
        close (fd[WRITE]); /* Chiude l'estremità originale usata
        execlp (argv[1], argv[1], NULL); /* Esegue il programma
        perror ("connect"); /* Non dovrebbe essere eseguita */
    }
    else { /* Figlio, lettore */
        close (fd[WRITE]); /* Chiude l'estremità non usata */
        dup2 (fd[READ], 0); /* Duplica l'estremità usata allo st
        close (fd[READ]); /* Chiude l'estremità originale usata
        execlp (argv[2], argv[2], NULL); /* Esegue il programma
        perror ("connect"); /* Non dovrebbe essere eseguita */
    }
}
```

12.14 Implementazione della pipeline

In generale il processo padre:

1. crea tante pipe quanti sono i processi figli meno uno
2. crea tanti processi figli quanti sono i comandi da eseguire
3. chiude tutte le estremità delle pipe
4. attende la terminazione di ciascun processo figlio

Ciascun processo figlio:

1. Chiude le estremità delle pipe che non usa
2. imposta le estremità delle pipe che usa sugli opportuni canali standard
3. invoca una funzione della famiglia exec() per eseguire il proprio comando

Esercizi