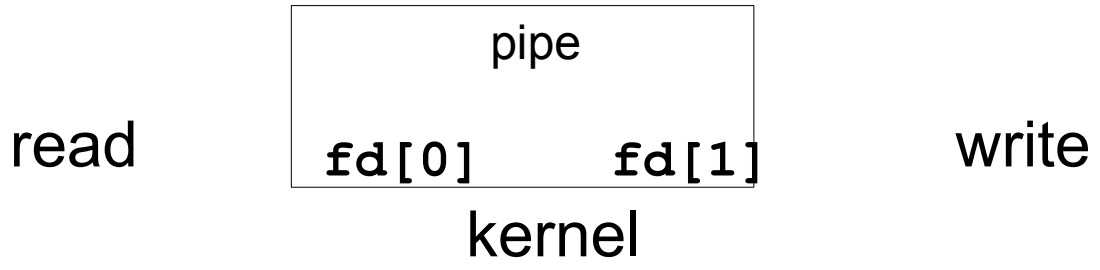
Syscall – Basic IPC

pipe (1/3)

- sono le più supportate strutture di Inter Process Communication (IPC) su sistemi *nix
- caratteristiche:
  - half-duplex
  - possono essere utilizzate solo tra processi che hanno un antenato comune nella gerarchia dei processi
- una pipe è formata da una coppia di file descriptor, il primo aperto in sola lettura, il secondo in sola scrittura; tutto ciò che viene scritto su quest'ultimo può essere letto dal primo



pipe (2/3)

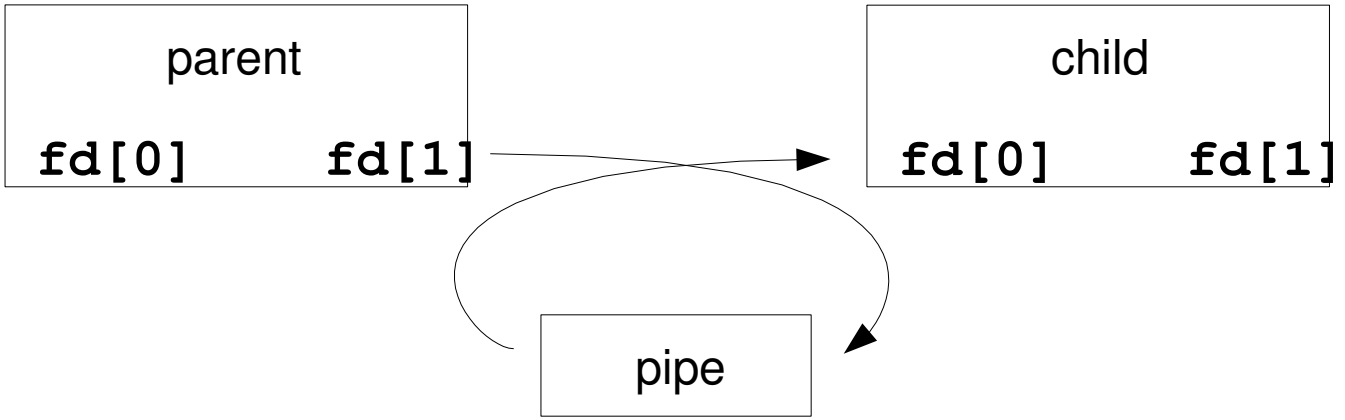
```
#include <unistd.h>

int pipe(int filedes[2]);
```

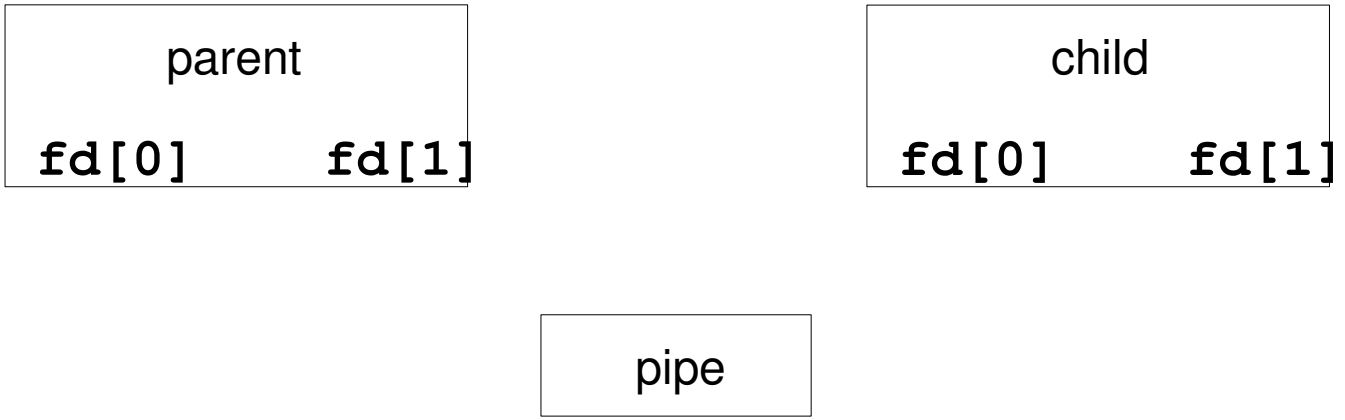
- la system call pipe permette di richiedere al kernel la creazione di una pipe
  - filedes: array di due file descriptor, viene riempito dalla syscall
    - fd[0] è aperto in lettura, fd[1] in scrittura
  - ritorna 0 in caso di successo, -1 altrimenti
- utilizzo tipico di pipe:
  1. creazione di una pipe
  2. fork
  3. uno dei due processi chiude fd[0], l'altro fd[1]

pipe (3/3)

- scenario dopo l'invocazione di pipe + fork



- scenario dopo l'invocazione di pipe + fork + close (assumiamo sia il figlio a voler scrivere al padre)



pipe – esempio

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

int main(void)
{
        int     n, fd[2];
        pid_t   pid;
        char    line[1024];
        if (pipe(fd) < 0) {
            fprintf(stderr, "pipe error\n");
            exit(1);
   }
   if ( (pid = fork()) < 0) {
            fprintf(stderr, "fork error\n");
            exit(1);
   } else if (pid > 0) {/* parent */
            close(fd[0]);
            write(fd[1], "hello world\n", 12);
   } else {     /* child */
            close(fd[1]);
            n = read(fd[0], line, 1024);
            write(STDOUT_FILENO, line, n);
        }
        exit(0);
}
```

popen / pclose

```
#include <stdio.h>

FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
```

- la libreria standard offre le funzioni popen/pclose per semplificare l'uso di pipe
  - popen = pipe + fork + exec + close
    - command: path del comando da eseguire
    - type: “r” per apire una pipe in lettura, “w” in scrittura
    - ritorna FILE pointer in caso di successo, NULL altrimenti
  - pclose chiude lo stream ed attende la terminazione
    - stream: stream da chiudere
    - ritorna stato di terminazione del proesso, -1 in caso di errori

mkfifo (1/3)

```
#include <sys/types.h>
#include <sys/stat.h>

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

- le FIFO (o *named pipe*) sono concettualmente simili alle pipe, ma:
  - esistono sul filesystem
  - permettono comunicazione tra processi arbitrari
- la syscall mkfifo permette di creare FIFO sul filesystem
  - pathname: percorso della FIFO sul filesystem
  - mode: flag come per open
  - ritorna 0 in caso di successo, -1 altrimenti

mkfifo (2/3)

- dopo la creazione le FIFO vengono utilizzate come file ordinari (open, write, close, ...)
- la semantica dell'utilizzo delle FIFO dipende dal flag O_NONBLOCK
  - nel caso in cui O_NONBLOCK non sia specificato
    - invocazioni di open in lettura restano bloccate fino a quando un altro processo non apre la FIFO in scrittura, e viceversa
  - nel caso in cui sia specificato
    - invocazioni di open in lettura ritornano immediatamente
    - invocazioni di open in scrittura falliscono con errno ENXIO se nessun processo ha ancora aperto la FIFO in lettura

mkfifo (3/3)

- rilevanza delle FIFO
  - shell scripting: comunicazione tra programmi che non compongono la stessa pipeline, senza ricorrere a file temporanei
    - il comando mkfifo (man **1** mkfifo) permette di creare FIFO da linea di comando
  - programmi con architettura client/server eseguiti su uno stesso host

- riferimenti
  - APUE, cap. 14, Interprocess Communication