Corso di Laboratorio di Sistemi Operativi
A.A. 2016–2017

Lezione 15

Ivan Scagnetto
ivan.scagnetto@uniud.it
Nicola Gigante
gigante.nicola@spes.uniud.it

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine

A.A. 2016–2017 - Primo Semestre - 30/11/2016

Comunicazione tra processi: pipes

Comunicazione tra processi: pipes

Affinchè due processi possano cooperare, è spesso necessario che **comunichino** fra loro dei dati.

- Una prima possibile soluzione a questo problema consiste nell'utilizzo condiviso dei file (e.g., leggendo e scrivendo in un file comune). Tuttavia tale approccio risulta inefficiente; inoltre vi è la possibilità che si verifichino dei problemi di contesa della risorsa condivisa.
- UNIX, per risolvere il problema, mette a disposizione una primitiva, detta **pipe**, che consiste in un canale **unidirezionale** di comunicazione che collega un processo ad un altro, generalizzando il concetto di file.
- È possibile inviare dati in una pipe attraverso la system call `write` e leggere dalla pipe attraverso la system call `read`, e in generale quasi tutte le funzioni di I/O su file.

Il **pipelining** di due comandi nella shell:

```
$ ls -l | less
```

è implementato tramite questo meccanismo.

Creare una pipe

Per creare una pipe, esiste l'apposita system call:

```
#include <unistd.h>
int pipe(int *filedes);
```

dove `filedes` deve puntare ad un array di due interi, che conterranno i file descriptor dei due capi della pipe:

- Il primo, `filedes[0]`, serve a leggere dalla pipe.
- Il secondo, `filedes[1]`, serve a scrivervi.
- I dati scritti da un capo vengono letti (in modo FIFO) dall'altro capo.

Passare i capi della papi ai processi figli

Perchè una pipe sia utile, è necessario in qualche modo passare uno dei due capi ad un processo figlio, in modo da poter comunicarci:

- I file descriptor restano aperti dopo la chiamata `fork()`: padre e figlio possono comunicare direttamente.
- È possibile **redirigere** un file descriptor aperto, ad esempio lo standard input, su un altro, ad esempio un capo della pipe. La redirezione resta in piedi dopo una chiamata a `execv()`.

Esempio

Creazione di una pipe

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

#define MSGSIZE 14

int main() {
  int pipes[2] = { };
  if(pipe(pipes) == -1) {
    perror("pipe call");
    return 1;
  }

  char msg[MSGSIZE] = { };
  pid_t pid = fork();
  switch(pid) {
    case -1:
      perror("fork call");
      return 2;
    case 0:            // processo figlio
      close(pipes[0]); // chiude l'altro capo
      write(pipes[1], "Hello, world!", MSGSIZE);
      break;
    default:           // processo padre
      close(pipes[1]); // chiude l'altro capo
      read(pipes[0], msg, MSGSIZE);
      printf("%s\n", msg);
      wait(NULL);
  }
  return 0;
}
```

Redirezione di un file descriptor

Per **redirigere** un file descriptor verso un altro è disponibile la chiamata di sistema `dup2()`:

```
#include <unistd.h>
int dup2(int fd1, int fd2);
```

Dopo la chiamata, il file descriptor `fd1` (che doveva essere già aperto) punterà alla stessa risorsa puntata da `fd2`.

Quindi questo codice:

```
int fds[2] = { };
pipe(fds);
dup2(fds[1], 1);
```

crea una pipe e ne collega il capo di scrittura allo standard output.

Esempi

Due esempi sono allegati alle slide:

- Il programma `lsgrep.c` esegue l'equivalente del comando:

```
$ ls -l | grep ⟨pattern⟩
```

  dove ⟨pattern⟩ viene dato da riga di comando.

- Il programma `guardieladri.c` implementa un semplice gioco in cui i due giocatori sono gestiti da due processi differenti, che comunicano tramite pipe.

Gioco guardie e ladri

Il programma `guardieladri.c` è un esempio di utilizzo delle pipe per coordinare e far comunicare più processi.

- Il gioco è chiamato "guardie e ladri":
  - il ladro (rappresentato dal carattere `$`) verrà mosso in modo casuale sullo schermo del terminale (80×24) dal computer
  - la guardia (rappresentata dal carattere #) verrà mossa dall'utente tramite i tasti freccia;
  - il gioco terminerà quando la guardia ed il ladro si incontrano.
- Nell'implementazione usiamo tre processi:
  - un processo principale, responsabile della visualizzazione e del controllo dell'evento in cui la guardia ed il ladro si incontrano
  - un processo figlio che aggiorna la posizione del ladro
  - un processo figlio che aggiorna la posizione della guardia
- Il programma utilizza la libreria **ncurses** (opzione `-lcurses`) per la gestione del terminale, usata da tutti i programmi testuali ma interattivi (es. editor di testo).

Esercizi

Esercizi

Per mercoledì 7 dicembre

1. Scrivere un programma che, dato il nome di un file sulla riga di comando, ne mostri il contenuto lanciando il comando 'cat' in modo equivalente al seguente comando da terminale:

```
$ cat < ⟨file⟩
```

   cioè redirigendo il file sullo standard input del processo figlio.

2. Scrivere un programma che, dato sulla riga di comando il nome di un file, esegua quanto segue:
   - Ogni riga del file sarà del formato:

```
comando1 arg1 ... argn | comando2 arg2 ... argn
```

   - Per ogni tale riga, il programma deve eseguire `comando1` e `comando2`, con relativi argomenti, collegando con una pipe lo standard output del primo allo standard input del secondo.