Quinta Esercitazione

Comunicazione tra processi Unix: pipe

System Call relative alle pipe

`pipe`	• `int pipe (int fd[])` **crea** una pipe e assegna **i 2 file descriptor relativi agli estremi** di lettura/scrittura ai primi due elementi dell'array fd. • Restituisce 0 in caso di creazione con successo, -1 in caso di errore
`close`	• Stessa system call usata per chiudere file descriptor di file regolari • Nel caso di pipe, usata da un processo per chiudere l'estremità della pipe che non usa.

Primitive di comunicazione

`read`	• Stessa system call usata per leggere file regolari, ma può essere **bloccante**: **Se la pipe è vuota: il processo chiamante attende fino a quando non ci sono dati disponibili.**
`write`	• Stessa system call usata per scrivere su file regolari, ma può essere **bloccante**: **Se la pipe è piena: il processo chiamante attende fino a quando non c'è spazio sufficiente per scrivere il messaggio.**
`dup`	**`fd1=dup(fd)`** crea una copia dell'elemento della tabella dei file aperti di indice **fd**. • La copia viene messa nella prima posizione libera (in ordine crescente di indice) della tabella dei file aperti. • Assegna a `fd1` l'indice della nuova copia, `-1` in caso di errore

Esempio – comunicazione tra processi mediante pipe

Realizzare un programma C che, utilizzando le *system call* di UNIX, preveda un'interfaccia del tipo:

`esame F1 F2.. FN PAROLA`

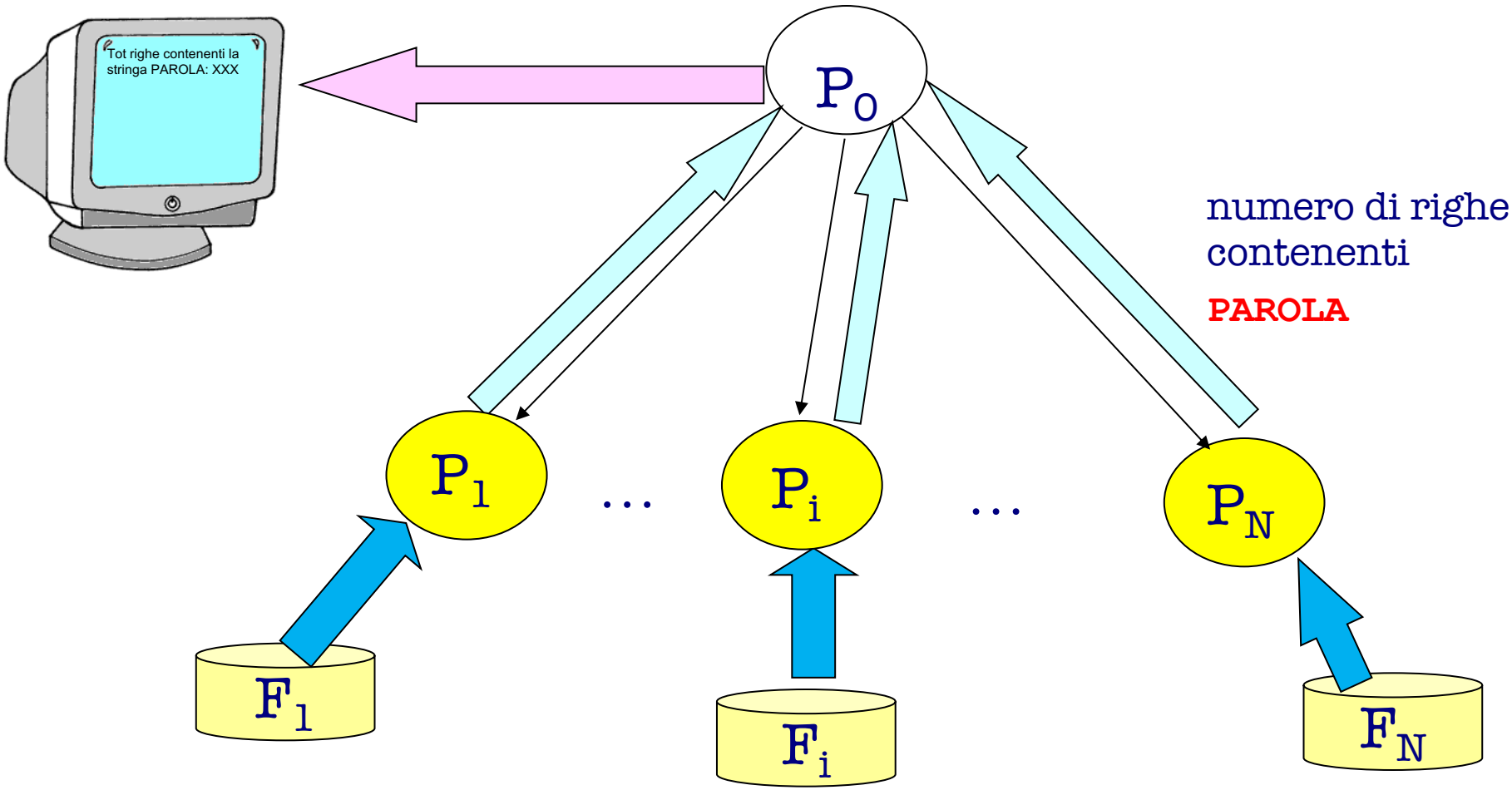
- **`F1, F2,.. FN`** rappresentano nomi assoluti di file
- **`PAROLA`** rappresenta una stringa

Il processo padre **P0** genera N figli (tanti quanti sono i filename dati come argomenti) P1, P2, .. PN:

Ogni figlio P_i, **tramite il comando `grep`**, deve **contare** le righe di $file_i$ nelle quali la stringa **PAROLA** compare almeno una volta**, e comunicare a P0 il valore risultante da tale conteggio.**

Il padre P0, sommerà i risultati ricevuti da tutti i figli, **stamperà sullo standard output il numero totale di righe contenenti almeno un'occorrenza** di **PAROLA,** e successivamente terminerà.

Esempio - Modello di soluzione



Esempio - Considerazioni

Ogni figlio Pi deve contare tramite l'esecuzione del comando **grep**. Per ottenere il numero di righe contenenti la stringa PAROLA, usare l'opzione **-c** (v. man grep).

Esempio di invocazione:

```
$ grep -c PAROLA FileX
```

Attenzione:

L'output di `grep -c C` è una **stringa** che rappresenta il numero di occorrenze di `C`. **Non è di tipo intero!**

Comunicazione figli->padre: quante pipe?

Il padre non ha bisogno distinguere il mittente di ogni messaggio ricevuto: il suo compito è sommare tutti i valori ricevuti → usiamo **1 pipe**.

Per ogni processo, l'output di grep deve essere inviato al padre P0 → ridirezione su pipe.

Soluzione dell'esercizio

```
#include <fcntl.h>
#include <stdio.h>
...
#define NP 8 // al massimo 8 figli
#define MAXS 256

void figlio(int fd_out, char filein[], char word[]);
void wait_child();

int pp[2]; // pipe per la comunicazione  figli ->padre
```

```
int main(int argc, char **argv){
   int pid[NP];
   int i,N; // numero di figli
   char parola[MAXS],buffer[MAXS];
   int tot_occ=0;
  // controllo argomenti:
  if (argc<3) // almeno 2 argomenti F1.. FN parola
  {  printf("sintassi! %s  F1.. FN parola\n", argv[0]);
     exit(-1);
  }
  N=argc-2; // N è il numero di file -> numero dei figli
  strcpy(parola, argv[argc-1]);
  if (N>NP)
    {   printf("troppi file !\n");
        exit(-2);
    }
```

```
// Apertura pipe pp:
if (pipe(pp)<0)
        exit(-3); /* apertura pipe fallita */

/* creazione figli: */
  for(i=0;i<N;i++)
  {   if ((pid[i]=fork())<0)
      {     perror("fork error");
            exit(-3);
      }
      else if (pid[i]==0) // figlio i
      {     close(pp[0]); //chiude il lato di lettura di pp
            figlio(pp[1], argv[i+1], parola);
      }
  }
```

```
// Padre:
close(pp[1]); // chiude il lato di scrittura di pp

for(i = 0; i < N; i++)
 {   char c;
     int letto, cont, fine,nread;
     cont=0;
     fine=0;
     while(!fine)
     {  nread=read(pp[0], &c, 1);// leggo il prossimo char dalla pipe
        if ((nread==1)&&(c!='\n')) // ho letto un char significativo
        {    buffer[cont] = c;
             cont++;
        }
        else {
            fine=1;
        }
     }
     buffer[cont]='\0';
     letto=atoi(buffer);
     tot_occ+=letto;
 }
```

```
   sprintf(buffer,"%d\n", tot_occ);
   write(1,buffer,strlen(buffer));// stampa il risultato
    // attesa figli:
    for(i = 0; i < N; i++)
    {   wait_child();
    }
    close(pp[0]); // chiudo la pipe
    exit(0);
} /* fine main */

void figlio(int fd_out, char filein[], char word[])
{  // ridirezione output su lato scrittura pipe:
  close(1);
  dup(fd_out);
  close(fd_out);
  // esecuzione grep:
  execl("/bin/grep", "grep", "-c", word, filein, (char*)0);
  perror("Execl fallita");
  exit(-1);
}
```

```
 void wait_child() {
  int pid_terminated,status;
  pid_terminated=wait(&status);
  if(WIFEXITED(status))
      printf("PADRE: terminazione volontaria del figlio %d con stato
 %d\n",
             pid_terminated, WEXITSTATUS(status));
  else if(WIFSIGNALED(status))
      printf("PADRE: terminazione involontaria del figlio %d a causa
 del segnale %d\n",
             pid_terminated,WTERMSIG(status));
}
```

Esercizio 1 (1/2)

Si realizzi un programma di sistema in C che preveda la seguente interfaccia:

```
./ese51 FileName Car
```

dove:

- **FileName** è il nome (assoluto o relativo) di un file di testo esistente nel file system.
- **Car** è un carattere

Si assuma che **FileName** contenga un testo di lunghezza arbitraria.

Il programma deve stampare sullo standard output il numero di righe di **FileName** che iniziano per **Car.**

A tal fine il processo P0 crea un figlio P1.

Esercizio 1 (2/2)

Il figlio P1 legge il contenuto di **FileName,** e invia a P0 le sole linee che iniziano per **Car**. Una volta conclusa questa operazione, P1 termina.

Il padre P0 conta il numero ***X*** di righe ricevute da P1 e stampa sullo standard output il messaggio:

"Il file <**FileName**> contiene <***X***> righe che iniziano per <**Car**>."

Successivamente P0 termina.

Note

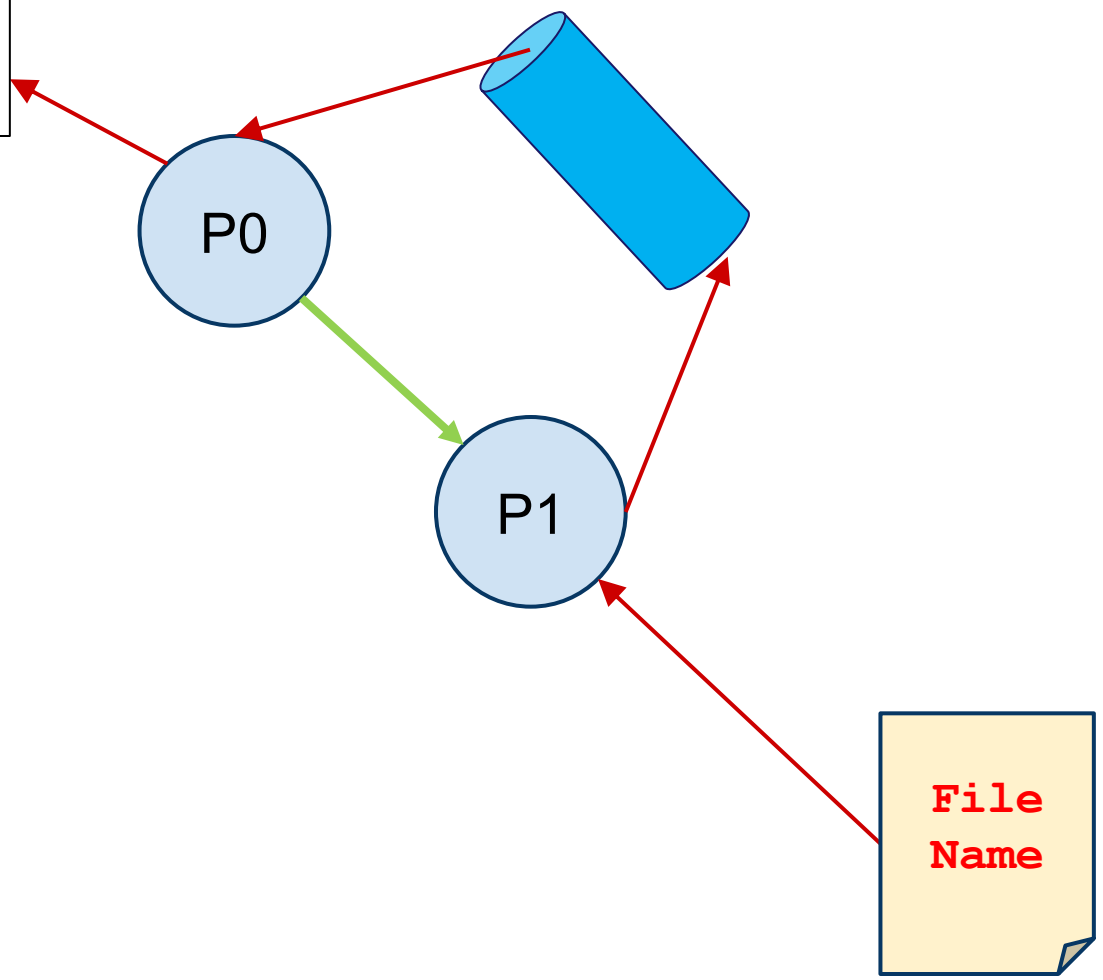
- **Come realizzare la comunicazione tra P1 e P0?**

  -> creazione di una pipe

Modello di soluzione

stdout

Il file <FileName> contiene X righe che iniziano per <Car>.



Pipe - Riflessioni

Le **pipe** sono uno strumento di comunicazione tra processi

- Consentono a processi in gerarchia di scambiarsi dati

Alcune differenze rispetto a read-write su file:

- **`read`** e **`write`** **bloccanti** (se la pipe è rispet. vuota o piena)
- La **`read`** ritorna zero se e solo se **tutti** i fd relativi al lato di scrittura sono chiusi

⇒Perché è importante NON LASCIARE APERTE ESTREMITA' INUTILIZZATE DELLE PIPE?

Le pipe possono essere uno **strumento di sincronizzazione** tra processi. **Quando conviene usare pipe e quando segnali?**

- Se devo comunicare dei dati tra processi, sono più comode le pipe,
- ma se un processo deve **fare delle operazioni intanto** che aspetta di ricevere qualcosa da un altro, **devo** ricorrere ai **segnali**!

Esercizio 2 – redirezione comandi

Come esercizio 5.1, ma:

- Il figlio `P1` realizza il filtraggio delle righe che iniziano per **Car** usando il comando **grep**.
  - Esempio: `grep ^p file1`

    filtra tutte le righe di `file1` che iniziano per il carattere p.

- il padre P0 conta le righe ricevute da `P1` usando l'opzione **-l** del comando **wc**

Modello di soluzione