Sesta esercitazione

Gestione Pipe Unix

Stefano Monti
stefano.monti6@unibo.it

Esercizio 1

Si realizzi un programma C usando le opportune system call Unix che abbia la seguente interfaccia

correggi2 <fileIn> <fileOut>

dove *fileIn*, *fileOut* sono due nomi (assoluti) di file.

In particolare

- *fileOut* : nome di file non presente su file system
- *fileIn* : file **binario** presente su file system contenente un numero N non noto a priori di triplette di interi:

	A	B	C	A	B	C	A	B	C	...
Ad esempio:	1	3	3	78	49	78	40	56	56	...

Specifiche

Il processo padre P0 deve generare due figli P1 e P2 (fratelli)

Il processo P2 deve

- **leggere** i primi due interi (A,B) di ogni tripletta in *fileIn*
- inviare a P1 il maggiore dei due interi (A o B)

Il processo P1 deve

- ricevere il maggiore dei due interi (A o B) di ciascuna tripletta
- leggere il valore di C e se questo risulta differente dal numero appena letto
  - **scrivere** il valore dell'intero maggiore **al posto del relativo elemento C** della tripletta
  - **comunicare** a P0 il valore corretto
- comunicare il termine della elaborazione (segnale esplicito??)

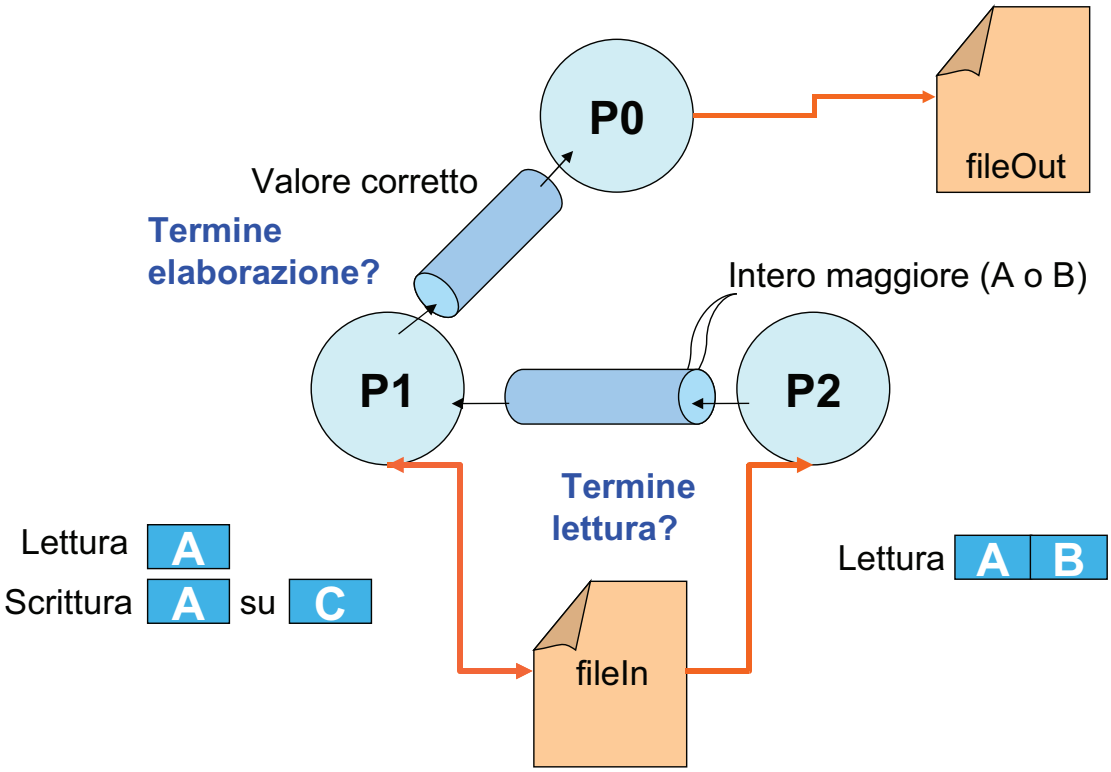
Il processo P0 deve

- sommare i valori ricevuti da P1
- al termine della elaborazione dei figli, **scrivere** tale valore su fileOut

Problematiche

- Gestione di file
  - Apertura in lettura o scrittura?
  - Apertura con IO pointer condiviso o separato?
- Gestione della comunicazione interprocesso
  - Quali (e quanti) strumenti?
  - PIPE e/o SEGNALI?
- Come gestire la sincronizzazione tra processi?
  - in particolare, terminazione della lettura/elaborazione
  - sono ancora necessari segnali?

Modello di soluzione




```
int pid0,pid1,pid2; int fileIn,fileOut;
int pipeP2P1[2],pipeP1P0[2];
int somma_correzioni=0;

int main(int argc, char* argv[]) {
      //--- controllo argomenti ------------
      ...
      //Apertura delle PIPE
      if (pipe(pipeP2P1)<0) {exit(-3);}
      if (pipe(pipeP1P0)<0) {exit(-3);}

      if((pid1=fork())<0) { perror("Errore nella fork:"); exit(-2);}
      if (pid1==0){//Codice figlio P1
            ...
      }
      else{
            if((pid2=fork())<0) { perror("Errore nella fork:"); exit(-2);     }
            if (pid2==0){//Codice figlio P2
                  ...
            }
            else{//Codice padre
                  ...
            }
      }
}
```

Codice P1

```
if (pid1==0){
        //Chiusura lati pipe non utilizzati
        close (pipeP2P1[1]); close (pipeP1P0[0]);

        //apertura fileIn con IO pointer separato: in lettura/scrittura
        if((fileIn=open(argv[1],O_RDWR))<0) {perror("Error opening file:");exit(-1);}
        int C, byteLetti; int elemento; //A o B

        while ((byteLetti=read(pipeP2P1[0],&elemento,sizeof(int)))>0){
            lseek(fileIn,2*sizeof(int),SEEK_CUR); //salto i due interi A e B
            read(fileIn,&C,sizeof(int)); //leggo C
            if (C!=elemento){
                lseek(fileIn, - sizeof(int),SEEK_CUR); //torno su C
                write(fileIn,&elemento,sizeof(int)); //scrivo su fileIn
                write(pipeP1P0[1],&elemento,sizeof(int)); //comunico al padre
            }
        }
        close(fileIn);
        close(pipeP2P1[0]);    close(pipeP1P0[1]);
    }
```

Quale è la condizione di uscita dal ciclo *while*?

Codice P2

```
if (pid2==0){//Codice figlio P2
        //chiusura lati pipe inutilizzati
        close(pipeP2P1[0]);    close(pipeP1P0[0]);    close(pipeP1P0[1]);

        //apertura fileIn con IO pointer separato
        if((fileIn=open(argv[1],O_RDONLY))<0) {
            perror("Error opening file:");
            exit(-1);
        }
        int coppia [2]; int byteLetti;
        while ((byteLetti=read(fileIn,coppia,2*sizeof(int)))>0){
            lseek(fileIn,1*sizeof(int),SEEK_CUR); //salto la lettura di C
            if (coppia[0]>coppia[1])
                write(pipeP2P1[1],&coppia[0],sizeof(int));
            else
                write(pipeP2P1[1],&coppia[1],sizeof(int));
        }
        //Chiusura delle risorse
        close(pipeP2P1[1]);    close(fileIn);
        exit(0);
    }
```

Codice P0

```
else{//Codice padre
        //chiusura lati pipe inutilizzati
        close(pipeP2P1[0]);   close(pipeP2P1[1]);   close(pipeP1P0[1]);
        if((fileOut=open(argv[2],O_WRONLY|O_CREAT,0600))<0) {
              perror("Error opening file:"); exit(-1);
        }
        int byteLetti,elemento;
        while ((byteLetti=read(pipeP1P0[0],&elemento,sizeof(int)))>0){
              somma_correzioni+=elemento;
        }
        char log[100];
        sprintf(log,"Somma dei valori corretti  : %d\n",somma_correzioni );
        write(fileOut,log,strlen(log));

        //Chiusura delle risorse
        close(fileOut);close(pipeP1P0[0]);

        //Raccolta stato terminazione dei figli
        wait(); wait();
  }
```


Alcune riflessioni

- Le pipe consentono di scambiare contenuto informativo
  - primitive di lettura/scrittura analoghe a file, tuttavia...
- Read e write
  - bloccanti (risp. se pipe vuota/piena)
  - lettura da pipe ritorna 0 solo se lato di lettura (tutti i fd relativi) è chiuso

  → è possibile usare le operazioni di lettura da pipe come elemento di sincronizzazione
- Cosa succede se dimentico di chiudere le pipe? (anche in altri processi??)

Esercizio 2

Si realizzi un comando in ambiente Unix, che, utilizzando le *system call* del sistema operativo, soddisfi le seguenti specifiche:

Sintassi di invocazione:

cerca filein1 filein2 stringa

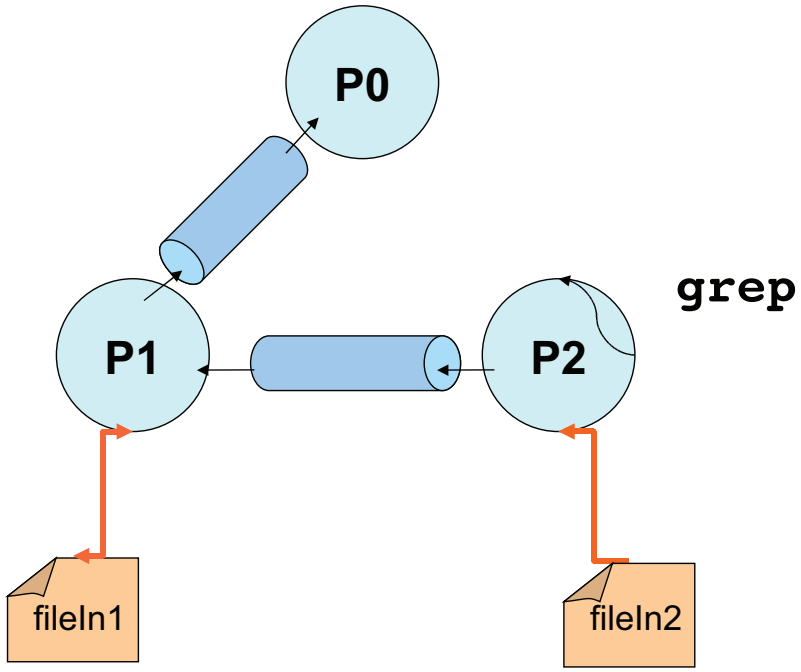
Significato degli argomenti

- **fileIn1 e fileIn2:** file di testo presenti nel filesystem e organizzati in righe di dim variabile (max 100 caratteri)
- **stringa:** stringa non nulla

Specifiche

- Il processo padre deve generare due figli (fratelli) P1 e P2
- P1 deve
  - leggere *fileIn1* e contare le righe in cui *stringa* compaia almeno una volta
  - ricevere da P2 il numero di righe contententi *stringa* in *fileIn2*
  - una volta ottenuti i due valori, comunicarli al padre
- P2 deve
  - leggere *fileIn2* e contare le righe in cui *stringa* compaia almeno una volta mediante comando **grep**
  - comunicare tale valore a P1 mediante opportuna ridirezione
- P0 deve
  - stampare a video il numero di occorrenze di *stringa* trovate in *fileIn1* e *fileIn2*

Schema di soluzione



Suggerimenti

- P2 deve contare mediante comando **grep**
  - `grep -c stringa fileIn2`
  - ridirezione output (piping) mediante syscall **dup**
  - **Attenzione**: che tipo di output per grep? (caratteri? valori binari?)

- P1 deve contare le righe contenenti *stringa*
  - funzione di libreria <string.h>
  - ritorna **puntatore nullo** (**NULL**) in caso negativo...

```
#include <string.h>

char *strstr(const char *haystack, const char *needle);
```

Variante

- Si supponga di non avere a disposizione l'opzione `-c` per grep:
  - `grep stringa fileIn2` → stampa video delle stringhe
  - `wc -l` → conta linee
  - piping di grep e wc
- un ulteriore fratello P3 lancia `grep` in pipe verso P2
- P2 esegue `wc`