Esercitazione 4
Gruppo AK

Accesso a file in Unix

Complementi sui file:

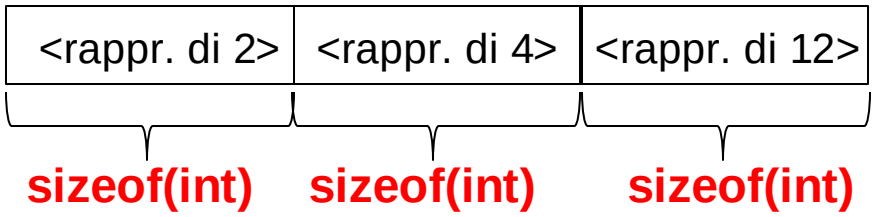
I file «**binari**»

File “Binari”

In Unix ogni file è una sequenza di bytes.
E’ possibile memorizzare all’interno di file la rappresentazione binaria di dati di qualunque tipo.

File Binario: ogni elemento del file è una sequenza di byte che contiene la rappresentazione binaria di un tipo di dato arbitrario.

Esempio:
file binario contenente la sequenza di interi [2,4,12]:



👉 Lettura di file binario contenente una sequenza di int:

```
int VAR;
read(fd,&VAR,sizeof(int)); //lettura del prossimo int
```

Come creare un File Binario?

Esempio:

file binario contenente una sequenza di interi da standard input:

```
#define dims 25
int VAR, k;
int fd;
char buff[dims]="";

fd=creat("premi", 0640);
printf("immetti una sequenza di interi (uno per riga),
terminata da ^D:\n"); // cntrl+D fornisce l'EOF a stdin
while (k=read(0, buff, dims)>0) {
     VAR=atoi(buff);
     write(fd, &VAR, sizeof(int));
}
close(fd);
```

Primitive fondamentali (1/2)

`open`	• Apre il file specificato e restituisce il suo file descriptor (fd) • Crea una nuova entry nella tabella dei file aperti di sistema (nuovo I/O pointer) • fd è l'indice dell'elemento che rappresenta il file aperto nella tabella dei file aperti del processo (contenuta nella user structure del processo) • possibili diversi flag di apertura, combinabili con OR bit a bit (operatore |)
`close`	• Chiude il file aperto • Libera il file descriptor nella tabella dei file aperti del processo • Eventualmente elimina elementi dalle tabelle di sistema
`unlink`	• Elimina il link al file specificato, cancellando pertanto il file (ritorna 0 se OK, altrimenti -1). • Occorre che il file descriptor sia stato chiuso per poterne eliminare il link.

Primitive fondamentali (2/2)

`read`	• **`read(fd, buff, n)`** legge al più n bytes a partire dalla posizione dell'I/O pointer e li memorizza in **`buff`** • Restituisce il numero di byte effettivamente letti 0 per end-of-file -1 in caso di errore
`write`	• **`write(fd, buff, n)`** scrive al più n bytes dal buffer **`buff`** nel file a partire dalla posizione dell'I/O pointer • Restituisce il numero di byte effettivamente scritti o -1 in caso di errore
`lseek`	• **`lseek(fd, offset, origine)`** sposta l'I/O pointer di **`offset`** posizioni rispetto all'origine. Possibili valori per origine: 0 per inizio del file (SEEK_SET) 1 per posizione corrente (SEEK_CUR) 2 per fine del file (SEEK_END)

Esercizio 1 (1/3)

Si realizzi un programma di sistema in C che effettua un'analisi a campione dei caratteri contenuti in due file di testo

Il programma deve prevedere la seguente sintassi di invocazione:

```
./analisi Fa Fb
```

- **`Fa`** ed **`Fb`** sono nomi assoluti di file **di testo** esistenti nel file system

Si assuma che **`Fa`** ed **`Fb`**:

- non contengano nessun **`'\n'`**
- abbiano la stessa lunghezza

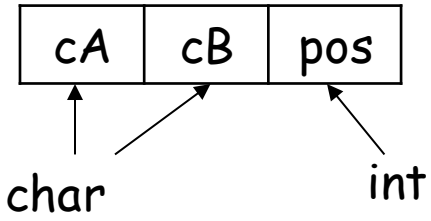
Esercizio 1 (2/3)

Il processo P0 per prima cosa genera due figli P1 e P2.

Poi P0 legge randomicamente e confronta i due file **`Fa`** ed **`Fb`** dall'inizio alla fine, ovvero esegue continuamente i seguenti step:

- genera un numero **r** tra 0 e 4.
- sposta i due I/O pointer avanti di **r** posizioni
- legge il carattere corrispondente da **`Fa`** e da **`Fb`**
- confronta i due caratteri letti

Per ogni differenza riscontrata, P0 deve salvare su un file **binario** temporaneo **`Fdiff`** i due caratteri (diversi) letti e la loro posizione



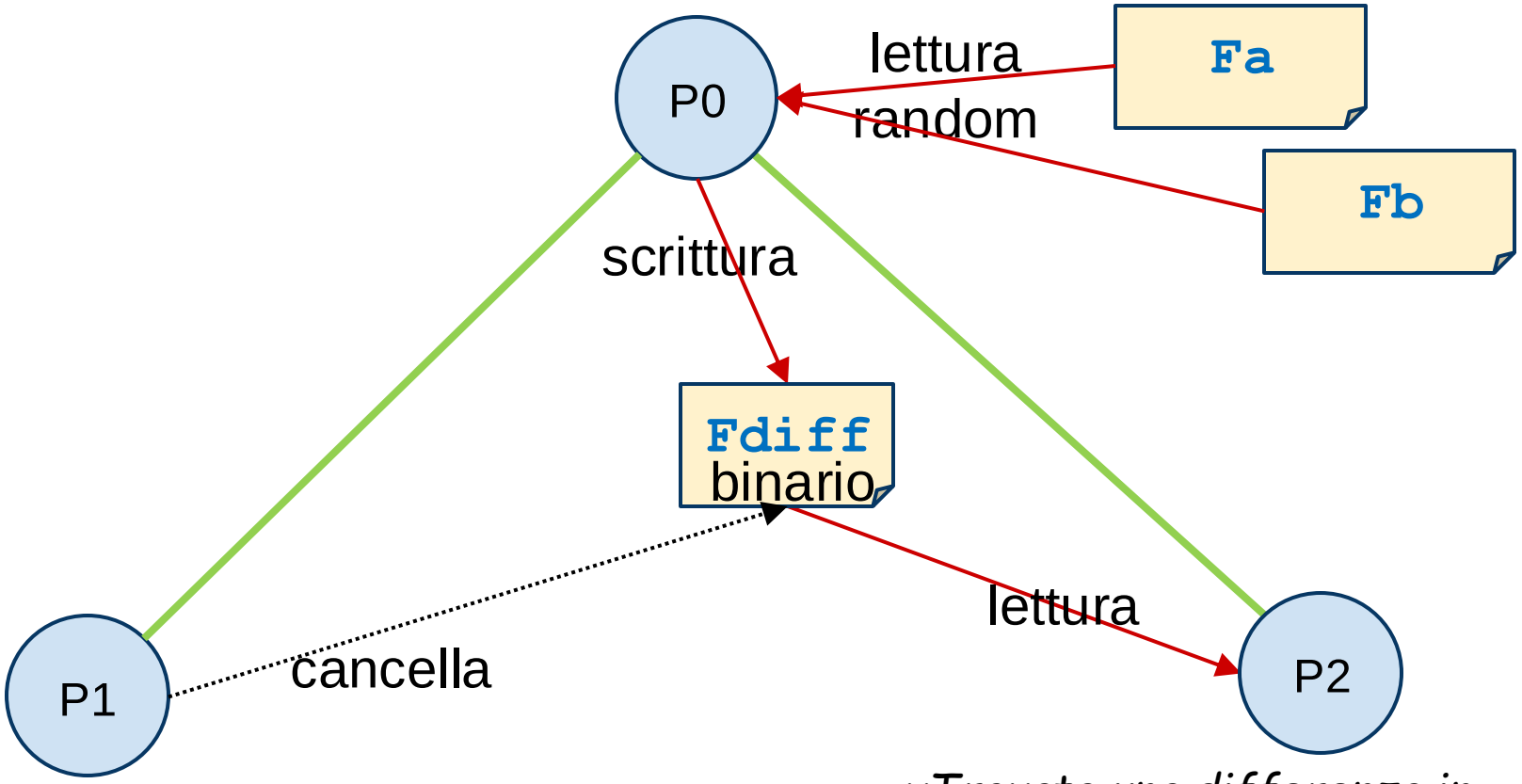
Esercizio 1 (3/3)

Il processo P2 deve leggere **`Fdiff`** e per ogni tripletta <***cA,cB,pos***> trovata deve stampare a video un messaggio dei tipo:

*«P2: Trovata una differenza in posizione **pos** :: cA=**cA** cB=**cB**»*

Una volta concluse tutte le operazioni, il processo P1 cancella il file temporaneo **`Fdiff`**

Modello di soluzione



Esercizio 1 – Riflessioni(1/2)

P0 deve leggere solo alcuni caratteri da **`Fa`** e **`Fb`** e saltare gli altri

➔ Quale primitiva?

P1 deve cancellare i file ➔Quale primitiva?

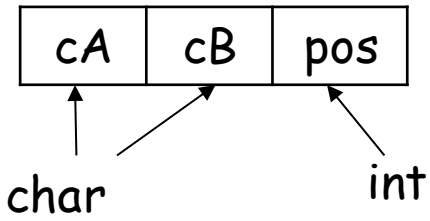
P0 deve creare subito P1 e P2, poi svolge il resto dei suoi compiti. Ordine necessario tra le operazioni:

- P0 crea il file **`Fdiff`**
- P2 legge **`Fdiff`** e stampa a video
- P1 cancella **`Fdiff`**

➔ i tre processi dovranno usare opportuni meccanismi per sincronizzarsi

Esercizio 1 – Riflessioni(2/2)

`Fdiff` deve contenere elementi del tipo:



Realizzabile in due modi (entrambi validi):

- scrittura/lettura di due char e poi di un int
- scrittura/lettura di una struttura:

```
typedef struct{
    char cA; //carattere letto da Fa
    char cB; //carattere letto da Fb
    int pos; //posizione
}elemento;
```

Esercizio 2 (1/3)

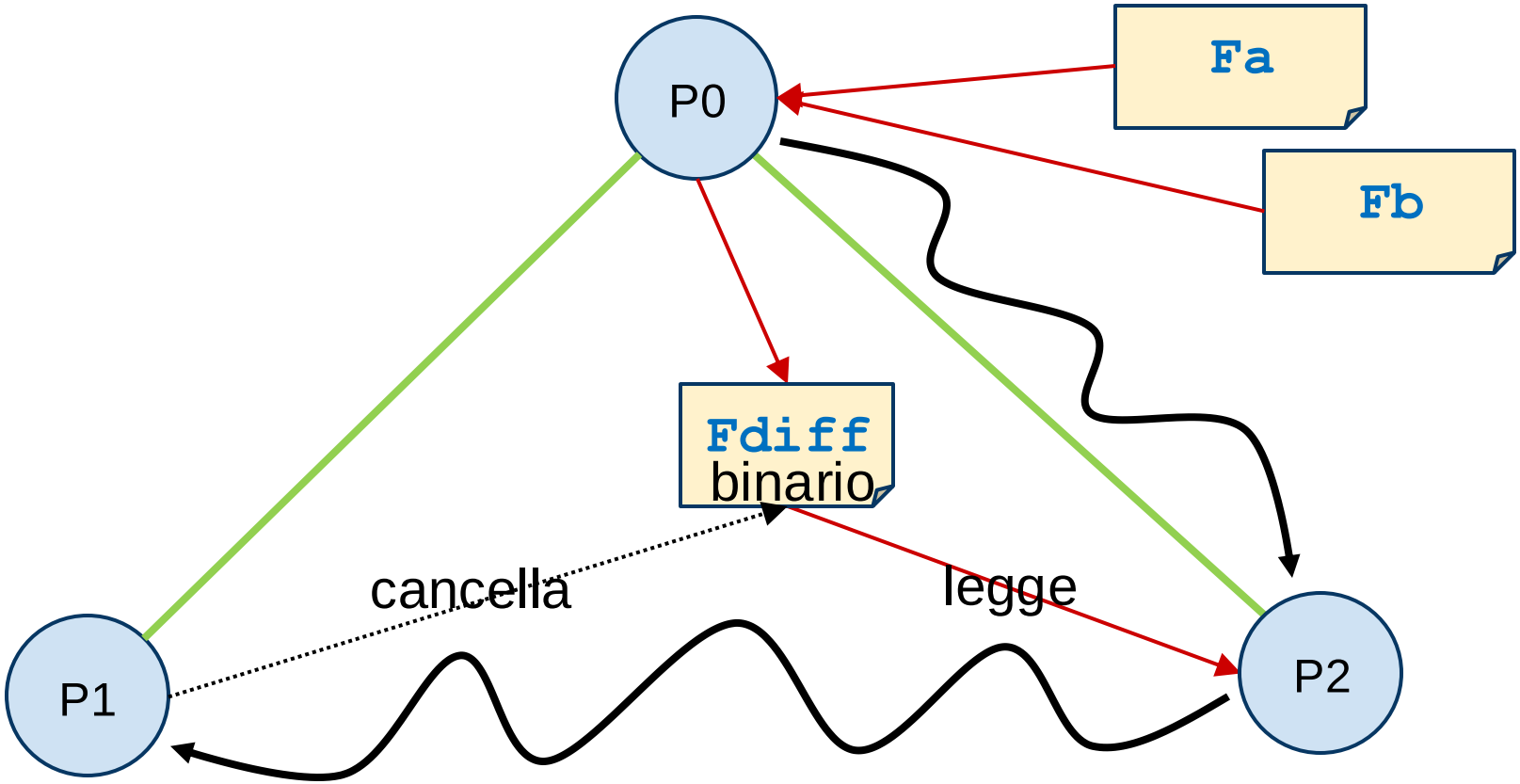
Si realizzi una variante dell'esercizio 1 in cui il file **`Fdiff`** è un file di testo.

Pertanto, per ogni differenza trovata tra **`Fa`** e **`Fb`**, il file **`Fdiff`** deve contenere una linea col seguente formato:

«cA cB pos»

Nota: per scrivere su file di testo è consentito usare solo la system call write(). NON è consentito usare fprintf().

Riflessione generale



Avrei potuto invertire i ruoli di P1 e P2?