

# **Esercitazione 4**

## **Gruppo LZ**

### **Accesso a file in Unix**

---

Complementi sui file:

I file «**binari**»

---

# File “Binari”

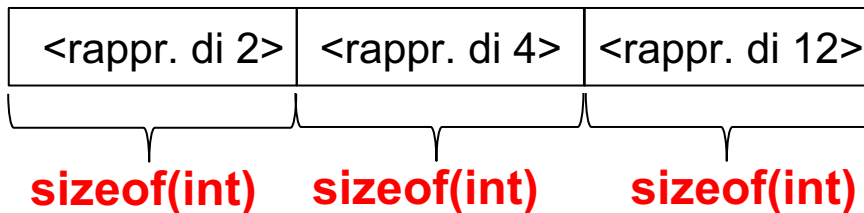
In Unix ogni file è una sequenza di bytes.

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)

<code>open</code>	<ul style="list-style-type: none"><li>• Apre il file specificato e restituisce il suo file descriptor (fd)</li><li>• Crea una nuova entry nella tabella dei file aperti di sistema (nuovo I/O pointer)</li><li>• fd è l'indice dell'elemento che rappresenta il file aperto nella tabella dei file aperti del processo (contenuta nella user structure del processo)</li><li>• possibili diversi flag di apertura, combinabili con OR bit a bit (operatore   )</li></ul>
<code>close</code>	<ul style="list-style-type: none"><li>• Chiude il file aperto</li><li>• Libera il file descriptor nella tabella dei file aperti del processo</li><li>• Eventualmente elimina elementi dalle tabelle di sistema</li></ul>
<code>unlink</code>	<ul style="list-style-type: none"><li>• Elimina il link al file specificato, cancellando pertanto il file (ritorna 0 se OK, altrimenti -1).</li><li>• Occorre che il file descriptor sia stato chiuso per poterne eliminare il link.</li></ul>

# Primitive fondamentali (2/2)

<b>read</b>	<ul style="list-style-type: none"><li>• <b>read(fd, buff, n)</b> legge al più n bytes a partire dalla posizione dell'I/O pointer e li memorizza in <b>buff</b></li><li>• Restituisce il numero di byte effettivamente letti 0 per end-of-file -1 in caso di errore</li></ul>
<b>write</b>	<ul style="list-style-type: none"><li>• <b>write(fd, buff, n)</b> scrive al più n bytes dal buffer <b>buff</b> nel file a partire dalla posizione dell'I/O pointer</li><li>• Restituisce il numero di byte effettivamente scritti o -1 in caso di errore</li></ul>
<b>lseek</b>	<ul style="list-style-type: none"><li>• <b>lseek(fd, offset, origine)</b> sposta l'I/O pointer di <b>offset</b> posizioni rispetto all'origine. Possibili valori per origine:<ul style="list-style-type: none"><li>0 per inizio del file (SEEK_SET)</li><li>1 per posizione corrente (SEEK_CUR)</li><li>2 per fine del file (SEEK_END)</li></ul></li></ul>

# Esercizio 1 (1/4)

Si realizzi un programma di sistema in C che *simula* un sistema di analisi dei consumi delle CPU di un PC con 3 core.

Il programma deve prevedere la seguente sintassi di invocazione:

**./cpu\_energy Ftemp Fout**

- **Ftemp** è il nome assoluto di un file **binario** non esistente
- **Fout** è il nome assoluto di un file **di testo** (esistente o non esistente)

# Esercizio 1 (2/4)

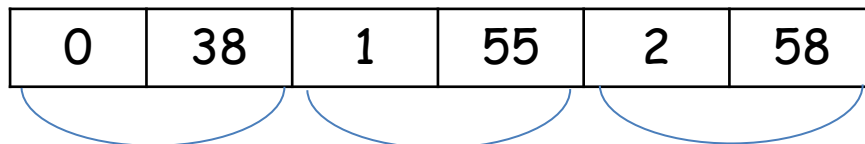
Il processo P0 genera un figlio P1 deputato alla creazione del file binario **Ftemp** riportante id e consumo di energia di ogni CPU.

Pertanto, P1 genera randomicamente 3 interi compresi tra 0 e 100 (ciascuno indicante il consumo in Wh di un core) e scrive sul file binario **Ftemp** tali informazioni utilizzando una struct definita come segue:

```
typedef struct{  
    int id; //id intero del core (=0, 1 o 2)  
    int energy; //Wh rilevati  
}consumo;
```

I consumi delle CPU dovranno essere scritti in **Ftemp** in ordine di id. Esempio di file Ftemp (**binario!**):

0	38	1	55	2	58
---	----	---	----	---	----





# Esercizio 1 (3/4)

Infine P1 deve rilevare quale dei due core è risultato più energivoro e segnarlo a P0.

il processo P0 deve compilare un file di output **Fout** a seconda di quanto segnalato da P1.

Ed in particolare deve:

- Leggere da Ftemp **SOLO** le info relative al core più energivoro saltando le altre.
- Riportare tale info in **Fout**. Esempio di file **Fout**:

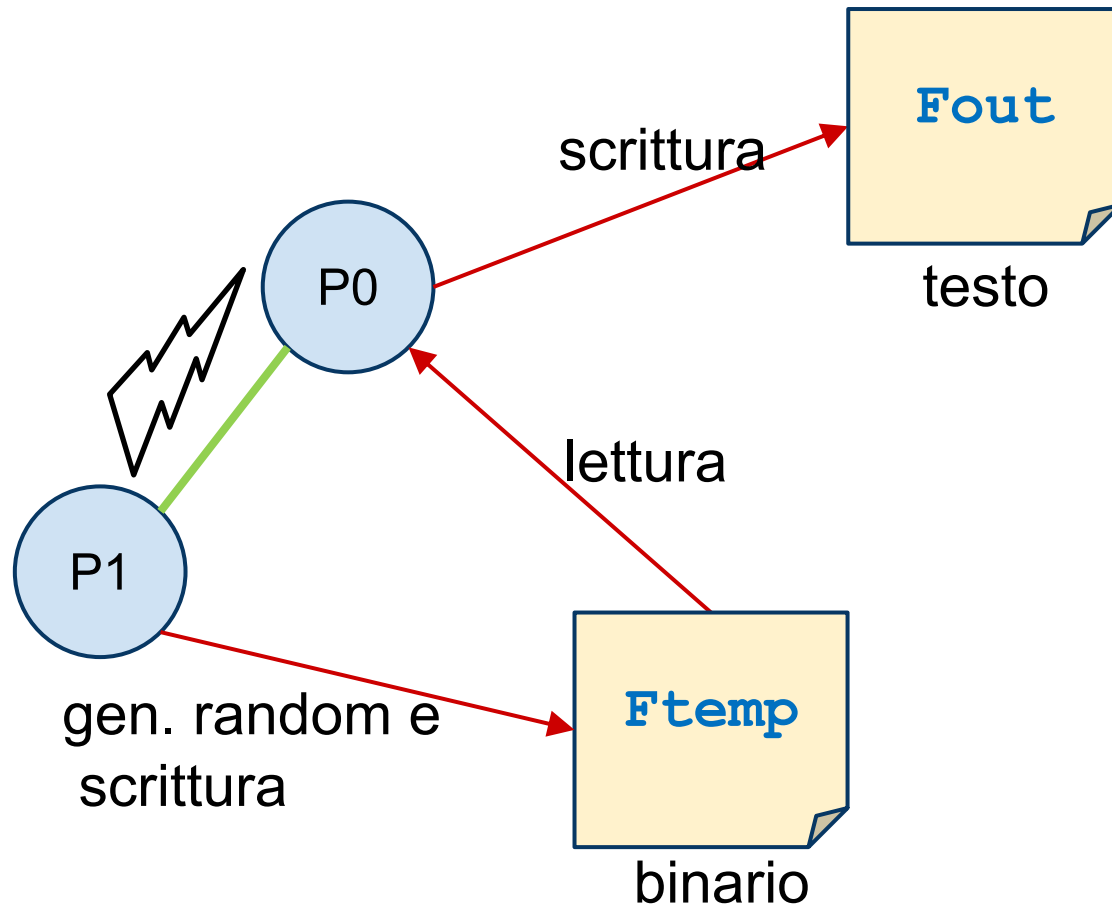
*core: 2*

*energy: 58*

Se il file **Fout** non esiste al momento dell'invocazione dello script, deve essere creato. Se esiste già, il suo contenuto deve esser sovrascritto.

Infine P0 **cancella** il file **Ftemp**

# Modello di soluzione



# Esercizio 1 - Riflessioni

- P0 e P1 devono operare sullo stesso file Ftemp. Tuttavia, poiché P0 inizierà ad operare solo dopo avere ricevuto un segnale da P1, la sincronizzazione delle operazioni di scrittura e lettura è implicita nella richiesta dell'esercizio.
  - P1 deve segnalare a P0 il core più energivoro dei tre, ma un solo segnale non può portare questa informazione. Come fare?
  - P0 deve leggere da Ftemp solo il valore di un elemento «consumo».. Quale primitiva per "saltare" gli altri?
  - P0 deve cancellare Ftemp dopo averlo usato. Quale primitiva?
-

# Esercizio 2

Si realizzi un programma di sistema in C con la medesima interfaccia dell'esercizio 1, ma nel quale i tipi dei file **Ftemp** e **Fout** siano invertiti:

**./cpu\_energ2 Ftemp Fout**

- **Ftemp** è il nome assoluto di un file **di testo** non esistente
- **Fout** è il nome assoluto di un file **binario** (esistente o non esistente)

Pertanto, P1 deve creare (randomicamente) un file di testo in cui ciascuna riga riporta le info di uno specifico core separate da ";" e P0 deve creare un file binario riportante solo le info del core più energivoro

Esempio di **Ftemp** :

0;38

1;55

2;58

Esempio di **Fout** :

2	58
---	----

NB: E' possibile usare il programma *leggi\_binario.c* per controllare se e' stato creato un file binario corretto

# Note

Come leggere un file a ritroso? (ricorda: il metodo di accesso è sequenziale) -> uso di **lseek**!

- Per posizionare l'I/O pointer a fine file:

**lseek(fd\_in, 0, SEEK\_END);**

- Per spostare l'I/O pointer sul byte precedente:

**lseek(fd\_in, -1 , SEEK\_CUR);**

- Per spostare l'I/O pointer indietro di N byte:

**lseek(fd\_in, -N , SEEK\_CUR);**

---