Esercitazione 4

Accesso a file in Unix

Alcuni complementi su accesso a file:

- **lseek**
- file «**binari**»

lseek

Nonostante il metodo di accesso adottato in Unix sia sequenziale, è prevista una system call che permette di spostare arbitrariamente l'I/O pointer:

```
lseek(int fd, int offset,int origine);
```

dove:

- **fd** è il file descriptor del file
- **offset** è lo spostamento (in byte) rispetto all'origine
- **origine** può valere:
  - ✓ 0: inizio file **(SEEK_SET)**
  - ✓ 1: posizione corrente **(SEEK_CUR)**
  - ✓ 2 :fine file **(SEEK_END)**

👉 in caso di successo, lseek restituisce un intero positivo che rappresenta la nuova posizione.

👉 in caso di errore (e.g. posizionamento oltre i limiti del file) restituisce -1

Esempio: lseek

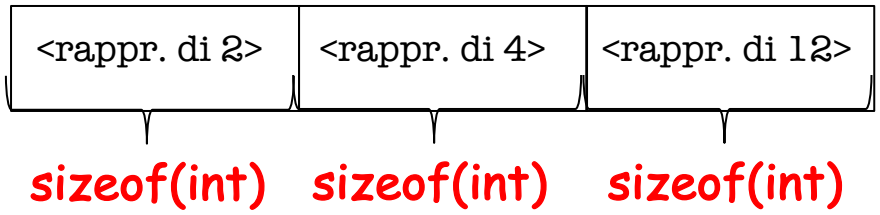
```
#include <fcntl.h>
main()
{int fd,n; char buf[100];
if(fd=open("/home/miofile",O_RDWR)<0)
   ...;
   lseek(fd,-3,2); /* posizionamento sul
      terz'ultimo byte del
                        file */

}
```

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 dati da standard input:

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

fd=creat("premi", 0777);
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);
```

Riassumendo: 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

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)

Esempio
(file di testo, file binari e lseek)

Esempio - Traccia (1/2)

Si realizzi un programma C che usi le opportune System Call Unix e realizzi la seguente interfaccia:

```
./filtra_e_inverti F FOut V
```

- **`F`**: è il nome di un file binario **esistente** nel filesystem, contenente una sequenza di interi .
- **`Fout`**: nome di un file di testo **non esistente** nel filesystem
- **`V`** è un intero

Esempio - Traccia (2/2)

Il programma deve realizzare il seguente comportamento:

Il processo padre **P0** genera **un figlio P1.**

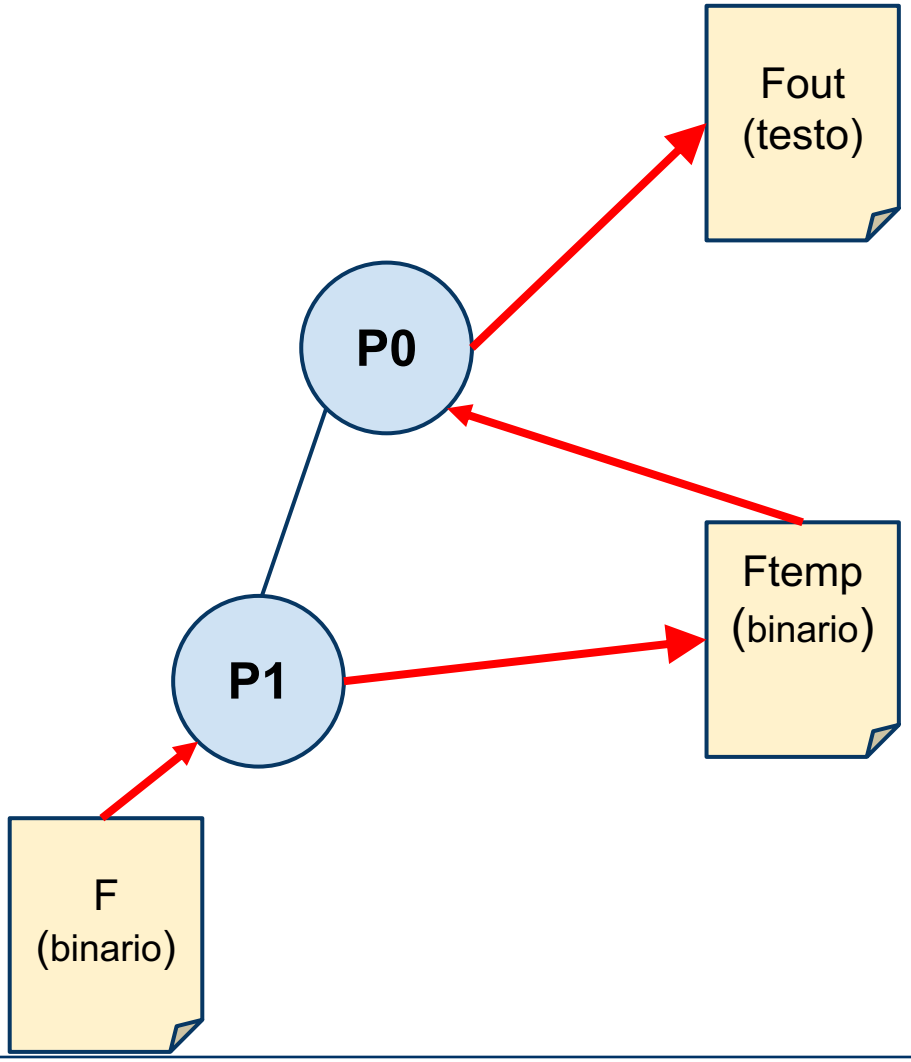
Comportamento del figlio P1:

1. a partire da F deve produrre un file binario temporaneo **"Ftemp"** che contenga i valori di F, ad esclusione delle eventuali occorrenze di V.
2. successivamente P1 terminerà.

P0 deve:

1. Attendere la terminazione del figlio P1.
2. A partire da **Ftemp, P0** dovrà scrivere nel **file di testo Fout** la sequenza dei valori contenuti in Ftemp **ordinata in ordine inverso, uno per riga.**
3. Prima di terminare, P0 dovrà **cancellare** Ftemp.

Esempio - Schema



Esempio

$ ./filtra_e_inverti F Pippo 4

Se il contenuto del file binario **F** è:

10	4	9	3	4	5	2

Il file binario **Ftemp** conterrà tutti i valori di F escluse le occorrenze di 4:

10	9	3	5	2

→ il file di testo **Pippo** conterrà:

2
5
3
9
10

Esempio - Problematiche

Sequenzializzazione tra:

- **scrittura di temp da parte di P1**
- **lettura di temp da parte di P0**

👉 uso di **wait/exit** per sincronizzare figlio e padre

Lettura di Ftemp a ritroso da parte di P0:

👉 una volta aperto in lettura Ftemp, uso di **lseek:**

- per posizionare **inizialmente** l'I/O pointer sull'ultimo intero del file ftemp:

```
lseek(fdtemp, -sizeof(int), SEEK_END);
```

- per **indietreggiare** di 2 interi dopo averne letto uno:

```
lseek(fdtemp, -2*sizeof(int), SEEK_CUR);
```

Esempio - Problematiche

Cancellazione di "temp" da parte di PO:
per cancellare un file si può usare la system call **unlink**:

```
int unlink(char *name);
```

dove:

- **name** è il nome del file
- ritorna 0, se OK, altrimenti -1.

👉 una volta chiuso il file **Ftemp**, si chiama la system call **unlink**:

```
unlink("Ftemp");
```

Esempio di soluzione (1/4)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#define DIM 25
void wait_child();

int main(int argc, char **argv)
{       int fd, fdout, fdtemp;
        int pid, n, V, K;
        char buff[DIM];
        //controllo argomenti:
        if (argc!=4)
        {       printf("errore argomenti \n");
                exit(-1);
        }
        else{
                if ((fd=open(argv[1], O_RDONLY))<0)
                {       perror("errore apertura F");
                        exit(-2);
                }
                close(fd);
        }
        V=atoi(argv[3]);
```

Esempio di soluzione (2/4)

```
pid=fork();
if (pid==0) //figlio
{       fd=open(argv[1], O_RDONLY);
        fdtemp=creat("Ftemp", 0777);

        while ((n=read(fd, &K, sizeof(int)))>0)
        {       if (K!=V)
                        write(fdtemp, &K, sizeof(int));
                else
                        printf("[figlio] scarto %d...\n", K);
        }
        close(fd);
        close(fdtemp);
        exit(0);
}
```

Esempio di soluzione (3/4)

```
    else { //padre
            wait_child();
            fd=creat(argv[2], 0777);
            fdtemp=open("Ftemp", O_RDONLY);
            // LETTURA A RITROSO di "Ftemp":
            lseek(fdtemp,-sizeof(int),SEEK_END);
            do{
                read(fdtemp, &K, sizeof(int));
                sprintf(buff, "%d\n", K);
                write(fd, buff,strlen(buff) );
            }while (lseek(fdtemp,-2*sizeof(int), SEEK_CUR)>=0) ;

            close(fdtemp);
            close(fd);
            unlink("Ftemp"); //cancellazione di "Ftemp"
            exit(0);

    }
} // fine main
```

Posiziona l'I/O pointer sull'ultimo intero del file Ftemp

legge il prossimo intero

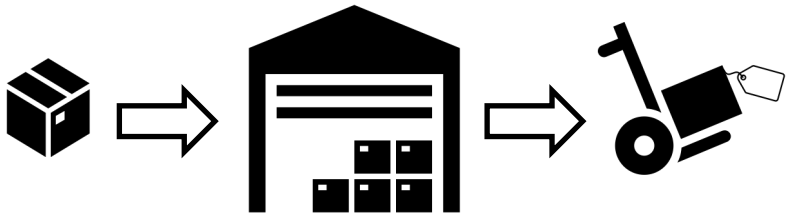
Posiziona l'I/O pointer sull'intero precedente all'ultimo letto

Esempio di soluzione (4/4)

```
void wait_child(){
    int status, pid_terminated=wait(&status);
    if ((char)status==0 && ((status>>8)!=0) )
        printf("P0: figlio con PID=%d term. volontariamente con
              stato %d.\n",pid_terminated, status>>8);
    else if ((char)status!=0)
        printf("P0: figlio con PID=%d terminato involontariamente
              per segnale %d\n",pid_terminated,(char)status);
}
```

Esercizio 1 (1/3)

Si realizzi un programma di sistema in C per la gestione del un magazzino di una fabbrica. Nel magazzino vengono stoccati i pezzi realizzati dalle varie linee di montaggio della fabbrica e dal medesimo magazzino i pezzi vengono prelevati per soddisfare gli ordini di vendita.



Il programma deve prevedere la seguente sintassi di invocazione:

```
./es41 Fin Fprodotti Fvendite
```

- **`Fin`** è il nome assoluto di un file di testo **esistente**, contenente i record dei pezzi prodotti e venduti in un determinato periodo di tempo.
- **`Fprodotti`** e **`Fvendite`** sono nomi assoluti di file binari **non esistenti**.

Esercizio 1 (2/3)

`Fin` contiene delle righe con il seguente formato:

<tipo>,<quant>,<op>

- **tipo** è un intero rappresentate il tipo dei pezzi
- **quant** indica la quantità di pezzi
- **op** è un carattere che indica la tipologia di operazione che ha coinvolto i pezzi (P=prodotti, V=venduti)

Esempio:

34,26,V → venduti 26 pezzi di tipo 34

12,18,P → prodotti 18 pezzi di tipo 12

123,2,P

…

Esercizio 1 (3/3)

Il programma deve popolare i file **Fprodotti** ed **Fvendite** usando le **informazioni** contenute in **Fin**:

- **nel file Fprodotti:** tipo e quantità dei pezzi prodotti letti da **Fin**;
- **nel file Fvendite:** tipo e quantità dei pezzi venduti letti da **Fin**.

Le informazioni da scrivere in **Fprodotti** e **Fvendite** saranno rappresentate ciascuna da una struct definita come segue:

```
typedef struct{
    int tipo;
    int quant;
}operazione;
```

Il programma dovrà scrivere in ognuno dei 2 file (**Fprodotti** e **Fvendite**) una sequenza **binaria** di dati di tipo **operazione**.

NB: per scrivere su un file binario dati di un qualunque tipo T:

```
T var;
int fd;
write(fd, &var, sizeof(T));
```

Esercizio 2 (1/3)

Si realizzi un programma di sistema in C per la gestione del magazzino.

Il programma da sviluppare prevede la seguente interfaccia:

```
./es42 Fprodotti Fvendite
```

dove:

- **`Fprodotti`** e **`Fvendite`** sono nomi di **file binari** esistenti contenenti le operazioni che hanno coinvolto i pezzi, ognuna rappresentata con una struttura di tipo **`operazione`** (v. esercizio1).

Il processo iniziale P0, dopo gli opportuni controlli sugli argomenti, dovrà creare 2 processi figli P1 e P2.

Esercizio 2 (2/3)

Comportamento di P1:

P1 dovrà leggere **a ritroso** il file **Fprodotti** e sommare le quantità in esso contenute per calcolare il numero totale ***Tp*** di pezzi *prodotti*.

Prima di terminare, P1 dovrà salvare il risultato del conteggio in un file di testo **./tempProd**

Comportamento di P2:

P2 dovrà leggere normalmente (dall'inizio alla fine) il file **Fvendite** e sommare le quantità in esso contenute per calcolare il numero totale ***Tv*** di pezzi *venduti*.

Prima di terminare, P2 dovrà salvare il risultato del conteggio in un file di testo **./tempVend**

Esercizio 2 (3/3)

Comportamento di PO:

Dopo aver creato i figli, PO attenderà la terminazione di entrambi, e confronterà quindi il contenuto di ./tempProd e ./tempVend.

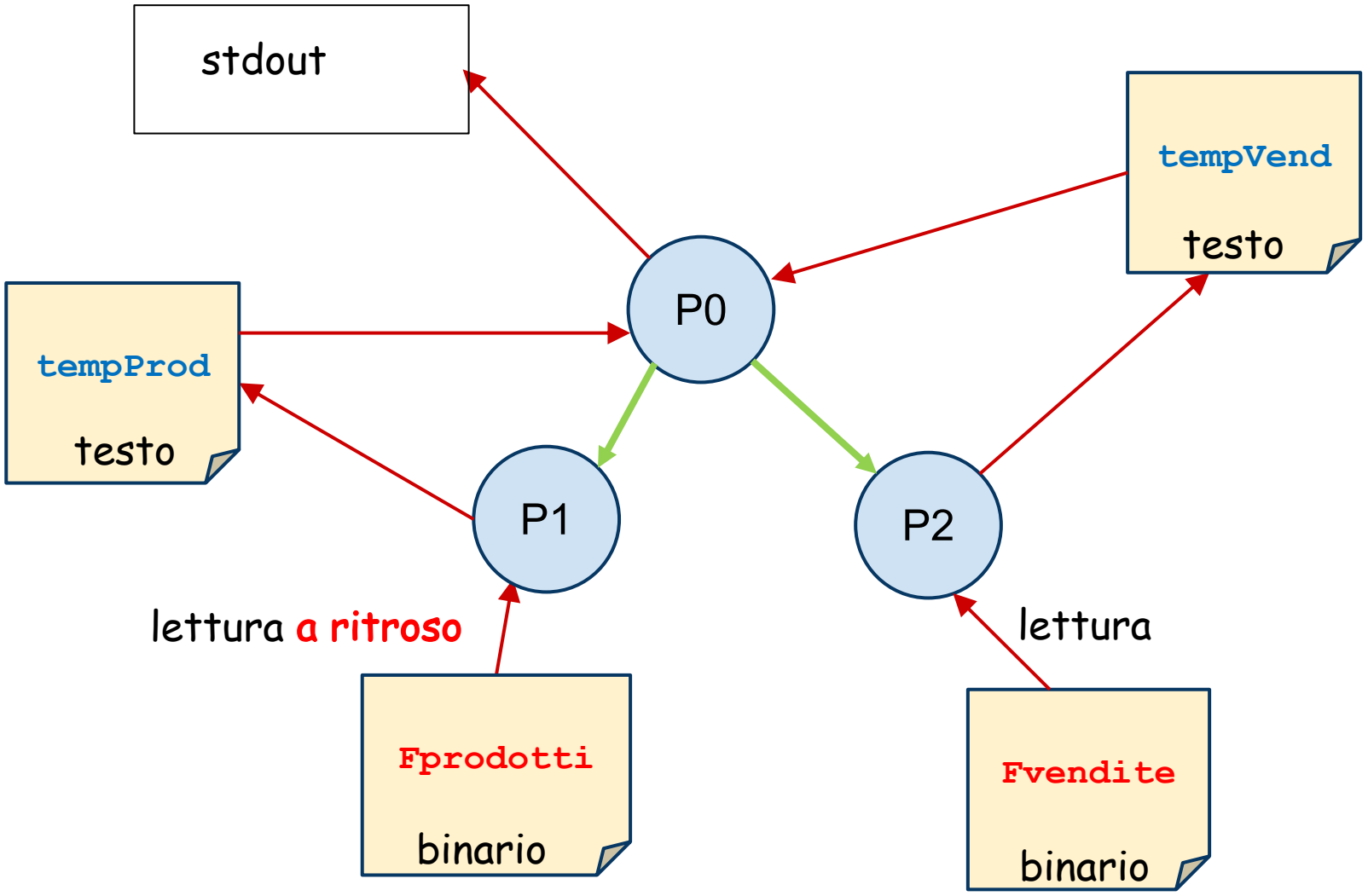
Se ***Tp>= Tv*** PO stamperà a video il messaggio:

I pezzi prodotti sono più di quelli venduti.

Se ***Tp<Tv*** PO stamperà a video il messaggio:

I pezzi prodotti sono meno di quelli venduti.

Modello di soluzione



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);
```