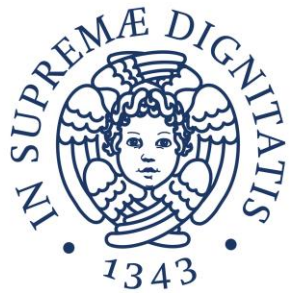


LABORATORIO DI SISTEMI OPERATIVI

Corso di Laurea in Ingegneria Informatica

A.A. 2023/2024

Ing. Maurizio Palmieri


maurizio.palmieri@unipi.it

ESERCITAZIONE 10

Virtual Filesystem

File descriptor e fork

Comunicazione fra processi mediante pipe

Esercitazione 10

Organizzazione del filesystem

- Filesystem Unix/Linux
- Collegamenti (link)
- Strutture e primitive di accesso ai file ordinari

Meccanismo delle pipe

- Comunicazione fra processi mediante pipe

Il filesystem in Unix/Linux

- I sistemi Unix sono caratterizzati dall'omogeneità
  - Ogni risorsa del sistema è rappresentata come un file
  - Esistono tre tipi di file
    - File ordinario
      Insieme di informazioni allocate in memoria di massa
    - File speciale
      Dispositivo fisico (periferica, porta di comunicazione…)
    - Directory
      Insieme di file

Il filesystem in Unix/Linux

- Una parte del disco è dedicata alla i-List, la lista di tutti i descrittori di file (i-node)
  - Ciascun i-node viene riferito e identificato mediante un i-number

- L'i-node descrive le caratteristiche del file:
  - Tipo — ordinario, speciale, directory
  - Info sui permessi — owner, group owner, permessi
  - Dimensione
  - Link — numero di nomi che riferiscono al file (hard links)
  - Vettori di indirizzamento — permettono di accedere al contenuto
  - …

Il filesystem in Unix/Linux

- Il contenuto di un "file direttorio" (directory) è costituito da una serie di record che descrivono il contenuto della directory:

  <nome_file1, i-number1>
  <nome_file2, i-number2>
  …

Il filesystem in Unix/Linux

- Quando viene creato un file di nome "fileName"
  - Viene creato il descrittore (i-node) e aggiunto alla i-List
    - Il nuovo i-node è identificato da un i-number IN
  - Al contenuto della directory viene aggiunto un nuovo record: <fileName, IN>

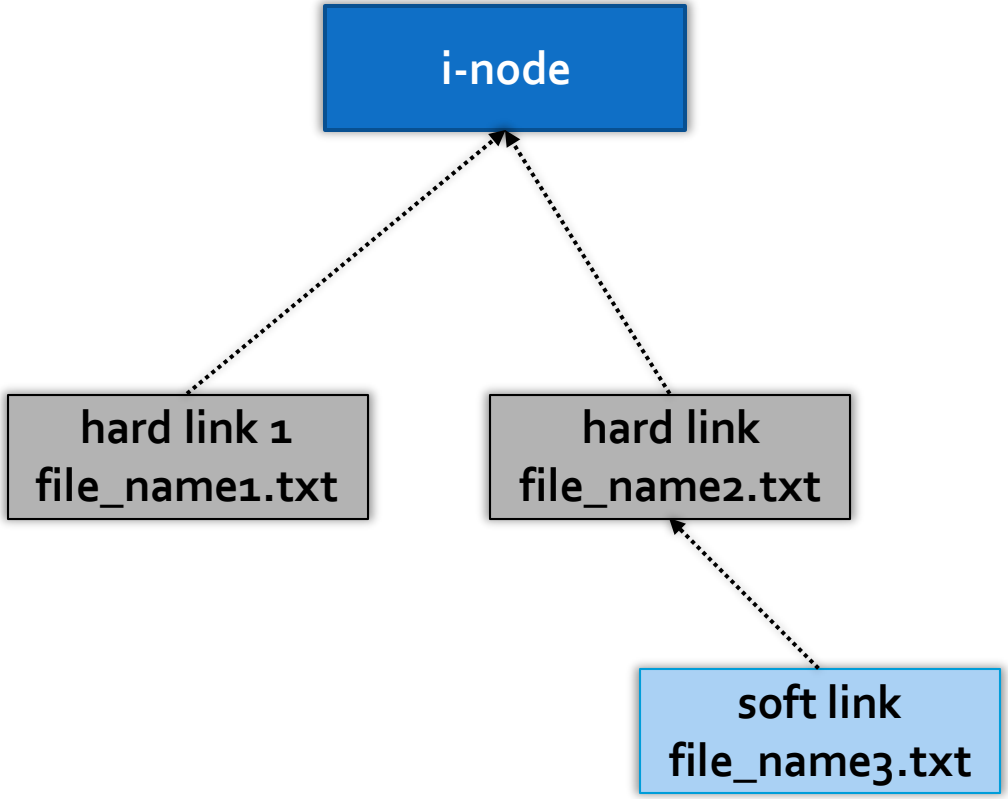
- Dopo la creazione, il file ha un solo nome ("fileName")
  - fileName è un hard link al descrittore del file

Collegamenti (link)

- Il filesystem permette di definire più hard link associati ad un i-node:
  - Sono nomi alternativi per lo stesso file

- E' possibile definire anche collegamenti simbolici (soft link)
  - E' un nome alternativo per fare riferimento ad un hard link

Collegamenti (link)



Hard e soft link – differenze

- Gli hard link puntano allo stesso i-node
  - Se elimino o sposto un hard link, questo non ha alcun effetto sugli altri hard link
  - Il descrittore del file (i-node) viene eliminato solo se il numero di hard link che vi fanno riferimento è zero

- Il soft link è un riferimento ad un hard link
  - Se elimino o sposto l'hard link, il soft link non è più in grado di accedere al file
  - L'eliminazione o lo spostamento di un soft link non ha alcun effetto sull'hard link

Comando ln

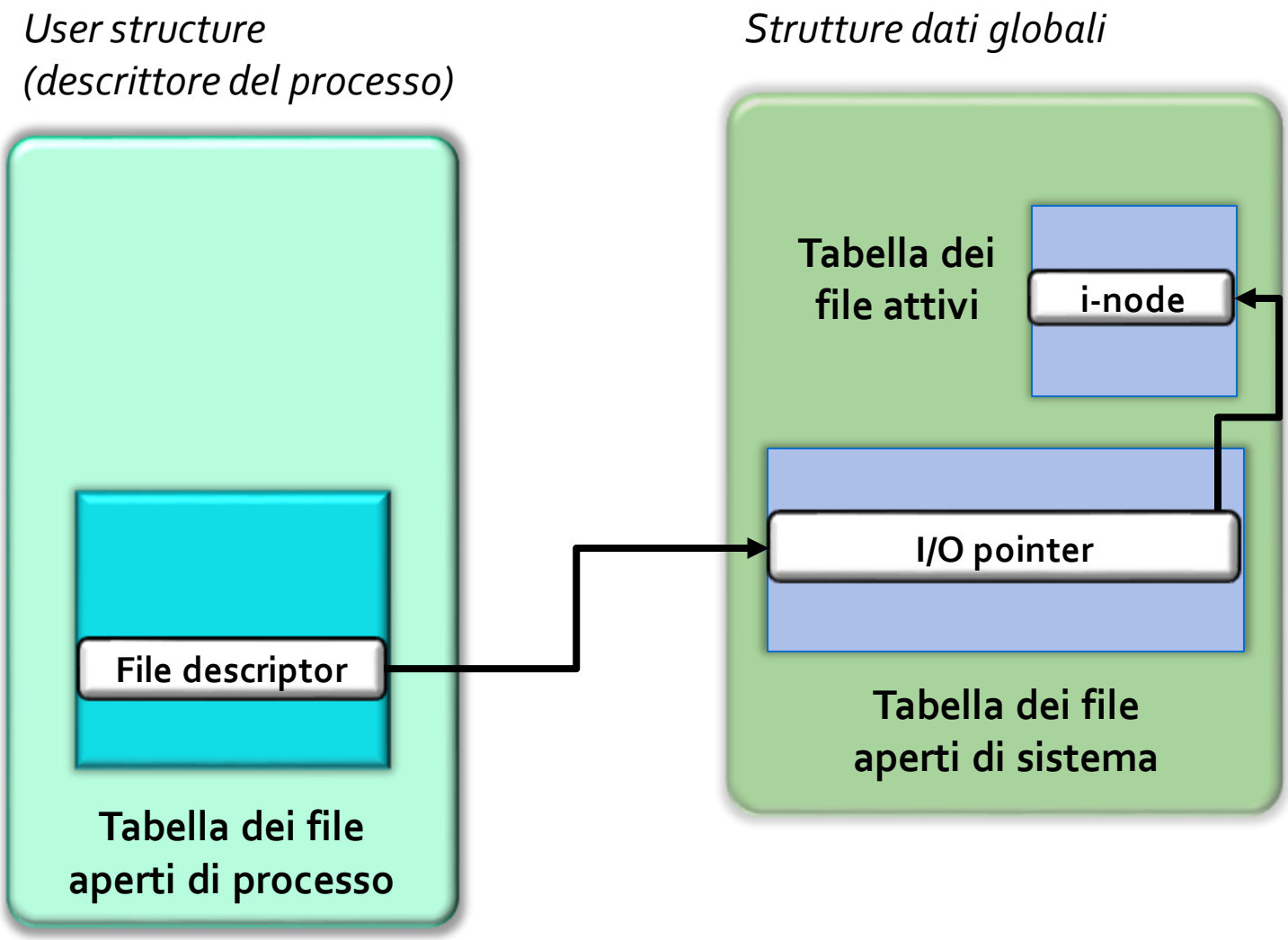
- Creazione hard link
- ln target link_name

- Creazione link simbolico (soft link)
- ln –s target link_name

Strutture dati per l'accesso ai file

- Il meccanismo adottato per l'accesso ai file è di tipo sequenziale:
  - Ad ogni file aperto è associato un I/O pointer, un riferimento per la lettura/scrittura sequenziale sul file
  - Le operazioni di lettura/scrittura provocano l'avanzamento del riferimento

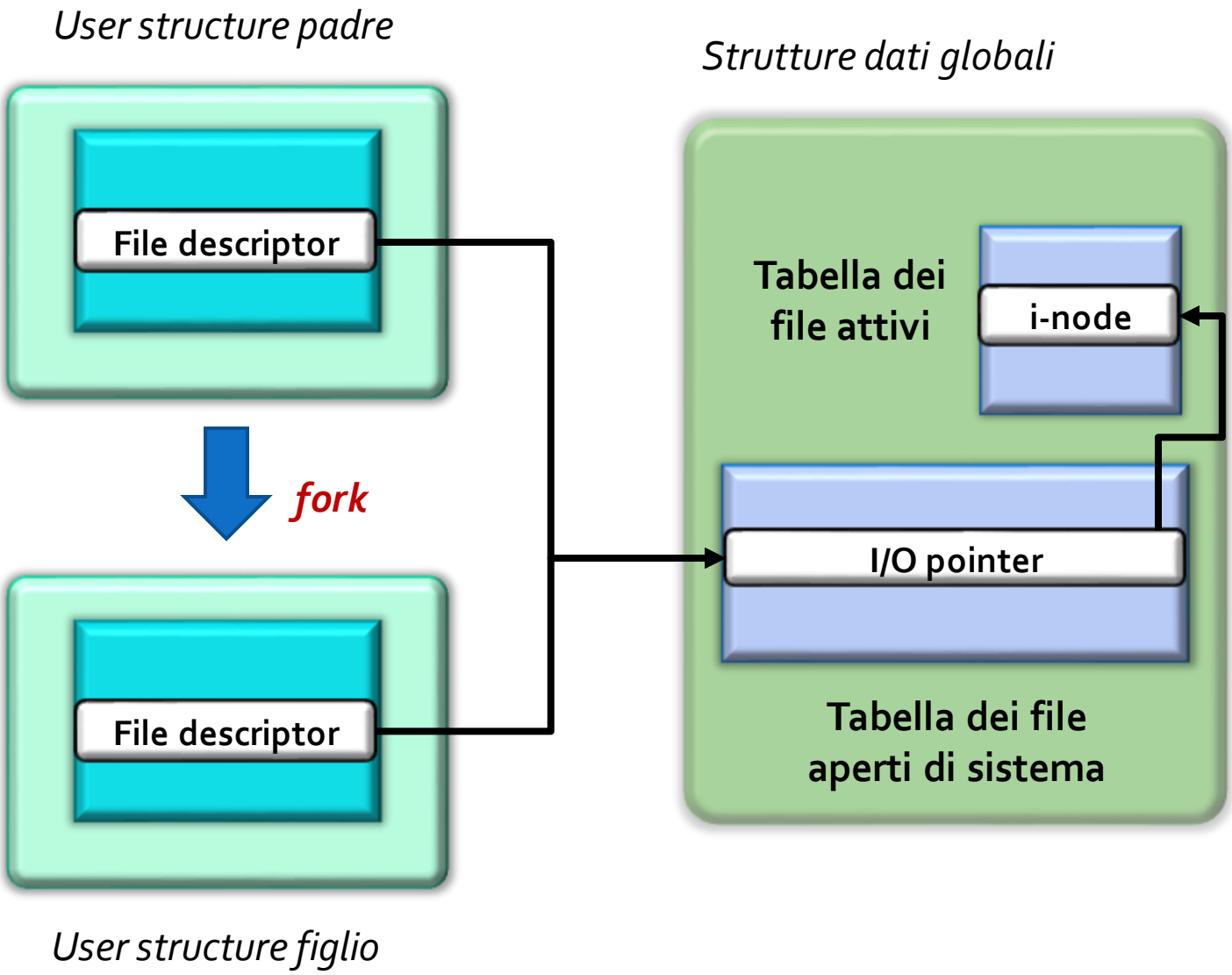
Strutture dati per l'accesso ai file



Strutture dati per l'accesso ai file

- Le strutture dati per l'accesso ai file sono gestite dal kernel:
- Tabella dei File Aperti di Processo
  - E' nella User Structure del processo
  - Ogni elemento (file descriptor) è un riferimento all'elemento corrispondente nella Tabella di File Aperti di Sistema
- Tabella dei File Aperti di Sistema
  - Contiene un elemento per ciascun file aperto dal sistema
    - Se due processi aprono lo stesso file allora ci saranno due entry separate
    - Ogni elemento contiene un I/O pointer al file e un riferimento all'i-node del file (che viene tenuto in memoria principale, nella Tabella dei File Attivi)
- I/O pointer e i-node permettono di trovare l'indirizzo fisico in cui effettuare la prossima lettura/scrittura sequenziale
- STDIN, STDOUT e STDERR sono descrittori di defaults
  - Vengono generati automaticamente al momento dell'esecuzione del programma

Strutture dati per l'accesso ai file



Strutture dati per l'accesso ai file

- Il processo figlio eredita dal padre una copia della User Structure, quindi anche una copia dei file descriptor
  - In questo caso, i due processi hanno descrittori che puntano allo stesso elemento della Tabella di File di Sistema, e quindi condividono l'I/O pointer nell'accesso sequenziale al file

Primitive per l'accesso ai file

- Apertura di un file descriptor

```
int open(const char* path, int flags)
```

  - `const char* path`
    Path del file da "aprire".
  - `int flags`
    Modalità di accesso. Ci sono varie macro definite <fcntl.h> per descrivere le possibili modalità. Se compatibili fra loro, più macro possono essere messe in OR. Esempi di macro:
    `O_RDONLY, O_WRONLY, O_RDWR.`
    Per la lista completa leggere "man 2 open".

  - Ritorna il file descriptor

- Dopo l'apertura, l'I/O pointer viene posizionato all'inizio del file se non è utilizzata la modalità `O_APPEND` (in tal caso, I/O parte dalla fine del file)

Primitive per l'accesso ai file

- Lettura da file

```
ssize_t read(int fd, void* buf, size_t count)
```

  - `int fd`
    Descrittore del file da cui leggere
  - `void* buf`
    Puntatore al buffer in cui scrivere i dati letti
  - `size_t count`
    Numero di byte da leggere (intero positivo)

  - Ritorna il numero di byte letti (valore negativo in caso di errore)

Primitive per l'accesso ai file

- Scrittura su file

```
ssize_t write(int fd, const void* buf, size_t count)
```

  - `int fd`
    Descrittore del file in cui scrivere
  - `void* buf`
    Puntatore al buffer da cui leggere i dati da scrivere nel file
  - `size_t count`
    Numero di byte da scrivere (intero positivo)

  - Ritorna il numero di byte scritti (valore negativo in caso di errore). Potrebbero essere meno di count, ad esempio, se è terminato lo spazio disponibile

Esempio (1/2)

- Esempio di lettura testo da file e stampa a video con buffer di dimensioni fisse:

```
#include<fcntl.h>
#include<stdlib.h>
#include<stdio.h>
#define BUF_SIZE 64

int main(int argc, char** argv) {
     if (argc < 2) {
              printf("Usage: %s FILENAME\n", argv[0]);
              exit(-1);
     }
     int fd = open(argv[1], O_RDONLY);
     if (fd < 0) {
              perror("Errore nella open\n");
              exit(-1);
     }
```

Esempio (2/2)

- Esempio di lettura testo da file e stampa a video con buffer di dimensioni fisse:

```
    char buffer[BUF_SIZE];
    ssize_t nread;
    while((nread=read(fd, buffer,BUF_SIZE-1)) > 0){
            buffer[nread] = '\0';
            printf("%s", buffer);
    }
    close(fd);
    if (nread < 0) {
            perror("Errore nella read\n"); exit(1);
    }
    exit(0);
}
```

Funzioni della libreria standard C

- open, read, write, close sono chiamate di sistema non bufferizzate
- La libreria standard del C mette a disposizione funzioni bufferizzate
  - fopen, fread, fwrite, flcose…
  - Alcune funzioni permettono la lettura/scrittura formattata (es. fscanf, fprintf…)

Comunicazione mediante scambio di messaggi – pipe

- I processi possono comunicare sfruttando il meccanismo delle pipe
  - Comunicazione indiretta, senza naming esplicito
  - Realizza il concetto di mailbox nella quale si possono accodare messaggi in modo FIFO
  - La pipe è un canale monodirezionale
    - Ci sono due estremi, uno per la lettura e uno per la scrittura

Comunicazione mediante scambio di messaggi – pipe

- Astrazione realizzata in modo omogeneo rispetto alla gestione dei file:
  - A ciascun estremo è associato un file descriptor
  - I problemi di sincronizzazione sono risolti dalle primitive read/write
    - Un lettore si blocca se la pipe è vuota
    - Uno scrittore si blocca se la pipe è piena
- I figli ereditano gli stessi file descriptor e possono utilizzarli per comunicare con il padre e gli altri figli
  - Per la comunicazione di processi che non si trovano nella stessa gerarchia si utilizzano altri strumenti, ad esempio i socket.
- Pagina del manuale:
  `man pipe`

Comunicazione mediante scambio di messaggi – pipe

- Creazione dei descrittori della pipe
  `int pipe(int fd[2])`

  - `int fd[2]`
    Vettore di due interi: conterrà i descrittori della pipe. Infatti, la funzione pipe salva in `fd[0]` l'estremo (il descrittore) della pipe per la lettura, in `fd[1]` l'estremo da usare per la scrittura

  - Ritorna zero se ha successo, -1 altrimenti.

ESERCIZI

Esercizio o

- Da terminale:
  - Creare un file "file1.txt"
  - Creare un hard link a file1.txt e chiamarlo hl.txt
  - Scrivere "Hello world" in file1.txt
    - Qual è il contenuto di hl.txt?
  - Abilitare i permessi in scrittura per "others" a hl.txt. Cosa succede ai permessi di file1.txt (verificare con ls –l)
  - Creare due link simbolici a hl.txt e chiamarli sym1.txt e sym2.txt
    - Eseguire ls –l e osservare l'ouput diverso in base al tipo di link
  - Eliminare file1.txt
    - Cosa succede a hl.txt?
  - Eliminare sym1.txt
    - Cosa succede a hl.txt?
  - Spostare o eliminare hl.txt
    - Cosa succede se si prova a vedere il contenuto di sym2.txt con il comando cat?

Esercizio 1 – file descriptor

- Da terminale, creare un file leggi.txt e scriverci "Hello world!"
- Scrivere un programma C in cui il main
  - Apre in lettura "leggi.txt" e salva il relativo descrittore in fd
  - Crea un processo figlio con la fork
  - Si sospende con la sleep per 3 secondi
- Il processo figlio:
  - Legge due caratteri dal file usando il descrittore fd e li stampa a video
- Il padre, dopo la sleep, legge da fd fino alla fine del file e stampa a video quello che ha letto
  - Notare gli effetti della condivisione dello stesso descrittore
- Adesso provare a chiudere il descrittore fd nel processo figlio, e ad aprirlo nuovamente. Come cambiano le cose?

Esercizio 2 – pipe

- Scrivere un programma C in cui il main
  - Crea una pipe
  - Crea un processo figlio
  - Si sospende per 3 secondi con la sleep
  - Scrive "Hello world\n" nella pipe
    (suggerimento: usare la funzione strlen definita in string.h per trovare la lunghezza della stringa al momento della write)

- Il processo figlio legge dalla pipe e stampa il messaggio che ha letto
  - Verificare il comportamento bloccante della read sulla pipe