# Files

Sistemi Operativi A.A: 2020-2021

Marco Tessarotto

# File descriptor (descrittore di file)

- Tutte le system call che effettuano I/O fanno riferimento ai file aperti usando un «file descriptor»: un numero intero >= 0
- I descrittori di file sono usati per riferirsi a tutti i tipi di file aperti (file regolari, directories,  speciali…)
- Ogni processo ha la propria lista di «file descriptor» (lista gestita dal kernel)
- Per convenzione, la maggior parte dei programmi di Unix si aspettano di avere già pronti, quando vengono avviati, tre file descrittori «standard»:
- 0: standard input, 1: standard output, 2: standard error
- Questi tre descrittori sono preparati dalla shell prima dell'esecuzione del programma (se il programma è lanciato da shell)

# fs ed i-node

- un **file system** («sistema di files»; abbreviato: **fs**) definisce e controlla il modo in cui i dati vengono memorizzati e recuperati (di solito su un dispositivo di archiviazione)
- L'**i-node** (nodo indice) è una struttura dati in un file system Unix che descrive un oggetto del file system (esempi: file regolari, directory).
- Ogni i-node contiene:
- gli **attributi** dell'oggetto
- le posizioni dei **blocchi** del disco dove si trovano i dati (i contenuti) dell'oggetto

# fs ed attributi, directory

- Timestamp: «istante», «orario»
- Gli **attributi** degli oggetti del file system includono **metadati** (timestamp: file change, file modify, file access), l'utente **proprietario** del file, i **permessi di accesso**

- Le directory sono elenchi di nomi assegnati agli i-node. Una directory contiene una voce per se stessa (dot), per il suo genitore (dot-dot) e per ciascuno dei suoi file «figli»

# Timestamp di un file

**ctime**

ctime is the i-node or **file change time**. The ctime gets updated when the file **attributes** are changed, like changing the owner, changing the permission or moving the file to another filesystem but will also be updated when you modify the content of a file.

**mtime**

mtime is the **file modify time**. The mtime gets updated when you modify a file. Whenever you update **content** of a file or save a file the mtime gets updated.

Most of the times ctime and mtime will be the same, unless only the file attributes are updated. In that case only the ctime gets updated.

**atime**

atime is the **file access time**. The atime gets updated when you **open** a file but also when a file is used for other operations like grep, sort, cat, head, tail and so on.

[oggi atime è modificato “un po’ di meno”; vedi anche https://lwn.net/Articles/244829/ ; https://unix.stackexchange.com/questions/238854/what-happened-to-proc-sys-fs-relatime-interval ]

http://web.archive.org/web/20160307172022/http://www.linux-faqs.info/general/difference-between-mtime-ctime-and-atime

# Visualizzare i timestamp di un file

- Mostrare mtime (file modify time)

ls –l <nome_file>

- Mostrare atime (file access time)

ls –lu <nome_file>

- Mostrare ctime (file change time)

ls –lc <nome_file>

- Oppure:

stat <nome_file>

# Visualizzare i timestamp di un file

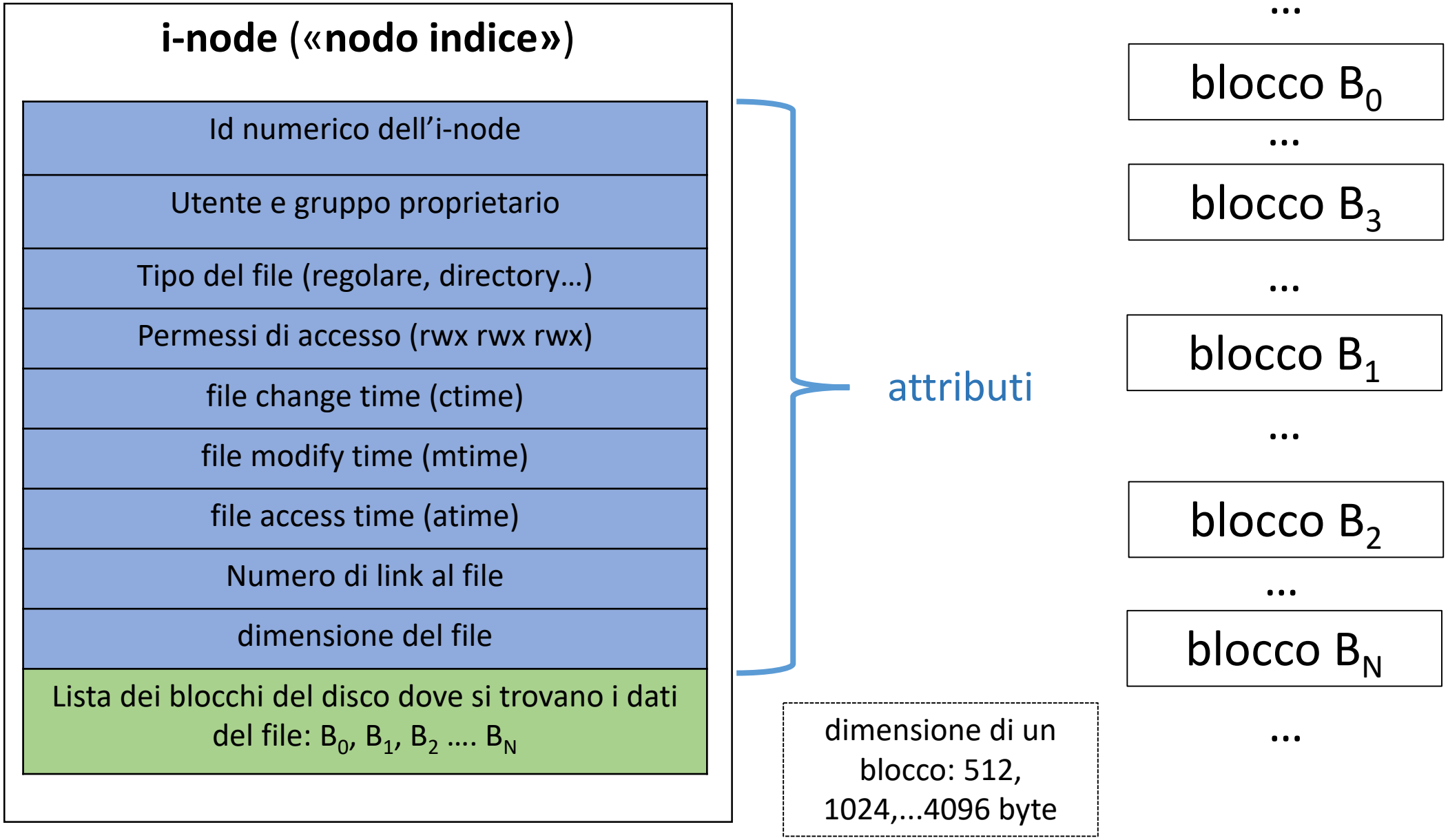
```
utente@pc:~$ stat prova.txt
  File: prova.txt
  Size: 6               Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 919529      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1003/ utente)   Gid: ( 1003/ utente)
Access: 2021-03-25 19:38:39.890640733 +0100     # il file è stato creato in quell'istante
Modify: 2021-03-25 19:38:39.890640733 +0100
Change: 2021-03-25 19:38:39.890640733 +0100
 Birth: -
utente@pc:~$ pico prova.txt # senza salvare, solo visualizzazione
utente@pc:~$ stat prova.txt
  File: prova.txt
  Size: 6               Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 919529      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1003/ utente)   Gid: ( 1003/ utente)
Access: 2021-03-26 10:17:16.580297955 +0100     # è stato invocato open() sul file
Modify: 2021-03-25 19:38:39.890640733 +0100
Change: 2021-03-25 19:38:39.890640733 +0100
 Birth: -
```

# Visualizzare i timestamp di un file

```
utente@pc:~$ chmod g-r prova.txt
utente@pc:~$ stat prova.txt
  File: prova.txt
  Size: 6               Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 919529      Links: 1
Access: (0624/-rw--w-r--)  Uid: ( 1003/ utente)   Gid: ( 1003/ utente)
Access: 2021-03-26 10:17:16.580297955 +0100
Modify: 2021-03-25 19:38:39.890640733 +0100
Change: 2021-03-26 10:22:38.456653928 +0100   # modifica attributi del file
 Birth: -


utente@pc:~$ pico prova.txt   # con modifica dei contenuti del file
utente@pc:~$ stat prova.txt
  File: prova.txt
  Size: 11              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 919529      Links: 1
Access: (0624/-rw--w-r--)  Uid: ( 1003/ utente)   Gid: ( 1003/ utente)
Access: 2021-03-26 10:24:02.248748097 +0100
Modify: 2021-03-26 10:24:09.044755806 +0100    # modifica contenuti del file
Change: 2021-03-26 10:24:09.044755806 +0100    # e viene cambiato anche ctime
 Birth: -
```

# File regolare



# Collegamento (link) tra filename e i-node

*utente@pc:~$ pico prova.txt*

*utente@pc:~$ ls -li prova.txt*

*-rw-rw-r-- 1 utente utente 6 mar 25 19:38 prova.txt*

***919529** -rw-rw-r-- 1 u140536 u140536 6 mar 25 19:38 prova.txt*

- 919529 è il numero dell'i-node
- è stabilito un "**hard link**" tra **filename** "prova.txt" (char *) e contenuti del file regolare (void *) stoccati nell'**i-node**
- Questa informazione è immagazzinata nella directory:

*utente@pc:~$ pwd*

*/home/utente*

# Tipi di files

- **File regolare**: «array di char»
- **Directory**: tabella a due colonne: filename, puntatore a i-node
- **Device**: un file speciale di dispositivo (device special file) è un'interfaccia per un driver di dispositivo (device driver: parte del kernel) che appare in un file system come se fosse un file normale (vedere /dev/*)
- **Pipe**: file speciale, è uno strumento per fare interoperare processi («tubo dati unidirezionale»); useremo nel corso
- **Socket**: «endpoint» per comunicazioni di rete
- **Symbolic links**: «soft link» (in contrapposizione a «hard link»)

| Nome del file (stringa di caratteri) | Riferimento a i-node |
|---|---|
| . | puntatore a i-node di questa directory |
| .. | puntatore a i-node della directory padre |
| «prova.txt» | ptr a i-node del file regolare |
| … | … |
| Debug | ptr a i-node della sub-directory |
| … | … |
| … | … |

# directory

**Ogni directory ha nella sua tabella almeno due elementi:**
**«dot» .**
**«dot-dot» ..**

**PID= 16445**
**In questo momento, il processo ha 7 file aperti**

**In verde: cosa «vede» il processo in user space**
**In azzurro: cosa «vede» in più il kernel**

# strutture dati del kernel per i file

- Il kernel gestisce le seguenti strutture dati relative ai file:
- Per-process **file descriptor table** (tabella dei descrittori aperti del processo; esiste una tabella per processo)
- System-wide **table of open file descriptions (open file table)**: tabella di sistema dei descrittori di file aperti (unica per tutto il sistema)
- The **file system i-node table**: la tabella degli i-node del file system

# file descriptor table (process)

- **Per ogni processo**, il kernel mantiene una tabella dei descrittori di file aperti (file descriptor table)

- Ogni riga della tabella contiene:

✓Numero del file descriptor (il numero di riga)

✓Insieme di flag (opzioni) che controllano il file descriptor

✓Il riferimento ad un «open file description» (in altra tabella)

Proc PID = 16445

| FD | | ptr to open file description |
|---|---|---|
| 0 | | … |
| 1 | | … |
| 2 | | … |
| 3 | | 42 |
| 5 | | … |
| 7 | | … |
| 10 | | 42 |
| | | |
| | | |
| | | |

# process **file descriptor table**

**Ogni processo ha la sua tabella**

**Esempio:**
**PID= 16445**
**In questo momento, il processo ha 7 file aperti**

**In verde: cosa «vede» il processo in user space**
**In azzurro: cosa «vede» in più il kernel**

# open file description (kernel)

- Il kernel mantiene, per l'uso da parte di tutto il sistema, una tabella di tutti gli «open file description» (chiamata anche **open file table**)
- [Kerrisk pag. 93-94]
- Una «open file description» contiene:
- File offset corrente (che viene aggiornato da read(), write(), lseek())
- Status flags (quelli passati a open())
- Modo di accesso: read-only, write-only, read-write
- Riferimento a i-node del file (Kerrisk cap. 14)

Proc PID = 16445

| FD | Fd flags* | ptr to open file description |
|---|---|---|
| 0 | | … |
| 1 | | … |
| 2 | | … |
| 3 | | 42 |
| … | | … |
| … | | … |
| 10 | | 42 |
| | | |
| | | |
| | | |

**Open file table (system-wide)**

| Row id | File offset | Status flags (pag. 74,75) | i-node ptr |
|---|---|---|---|
| | | | |
| | | | |
| 42 | 0x1234 | O_RDWR … | 0x73478234 |
| | | | |
| | | | |

Il processo ha vari file aperti, tra cui un file aperto con fd=3
successivamente ha usato dup() per duplicare fd=3 in fd = 10

quindi fd=3 e fd=10 puntano allo stesso descrittore di file aperto
i due fd condividono lo stesso file offset

* close-on-exec: FD_CLOEXEC

**Process file descriptor table**

# File system i-node table

- Ogni file system ha una tabella di i-node per tutti i file che si trovano nel file system stesso

- i-node di ogni file include:
- Tipo di file (regolare, directory,…) ed i permessi
- File size, timestamps
- Riferimento ai contenuti del file cioè ha le informazioni per accedervi

# Shell e descrittori di file

- La shell è il programma che accetta comandi
- La shell «interattiva» ha i suoi tre descrittori di file standard (0,1,2) collegati al terminale (dentro il quale la shell esegue)
- Quando chiediamo alla shell di eseguire un programma, questo riceve una copia dei descrittori della shell
- Se nel comando dato alla shell specifichiamo delle «redirezioni», la shell si occupa di modificare i descrittori da passare al programma da eseguire

# open() e file descriptor

**fd** = open(pathname, flags, mode)

- La system call open() apre il file specificato dal suo «pathname» (percorso nel file system) e restituisce un **file descriptor** (un int >= 0) che dovrà essere utilizzato nelle successive system call per operare con il file aperto
- ad esempio:

numread = read(**fd**, buffer, buffer_size)

- legge dal file (aperto in precedenza) identificato univocamente dal numero intero fd

# Allocazione dei file descriptor

- Se open() ha successo, deve restituire il minor file descriptor inutilizzato

- Esempio:
- se i file descriptor **0,1,2** sono già utilizzati ed il programma invoca open() per aprire un file (e open ha successo) allora open utilizzerà e restituirà il file descriptor **3**

# Esempio con file descriptor

- Altro esempio:
- i file descriptor 0,1,2 sono già utilizzati
- il programma chiude il file associato al fd 1 con l'istruzione:

close(1);

- il programma successivamente apre un file con open() che deve restituire il minor file descriptor inutilizzato, quindi restituisce:

File descriptor = 1

# System call ed atomicità

- Atomicità: ««l'atomo è indivisibile»»
- Kerrisk pag. 90
- Molte system call sono eseguite atomicamente

- Questo vuol dire che il kernel garantisce che tutti i passaggi che compongono una system call sono completati come se fossero una singola operazione, senza venire interrotti da un altro processo

Il kernel mantiene la lista dei processi attraverso una double linked list di PCB