

Syscall – File I/O

Digressione: man 2 ...

- Le funzioni della libreria standard UNIX associate alle system call di Linux sono documentate nella sezione 2 di man
 - e.g.: man 2 open
- Ogni manpage di system call riporta:
 - header: file .h che devono essere inclusi per avere accesso al prototipo delle funzioni
 - sinossi: possibili invocazioni C delle funzioni
 - descrizione del funzionamento e dei parametri attuali
 - valore di ritorno
 - errori: valori impostati per la variabile errno
 - conformità

Digressione: strace

- il programma strace è molto utile nel debugging di programmi al livello di astrazione delle system call
- invocazione: *strace* <programma> <arg> ...
 - il programma specificato viene eseguito
 - su stderr vengono stampate tutte le invocazioni di system call effettuate dal processo corrispondente, comprensive di valori di ritorno
 - vengono inoltre stampate le ricezioni di segnali
- riferimenti:
 - man strace

File descriptor

- ad ogni processo è associato un insieme di file aperti (/proc/<pid>/fd/*)
 - il kernel mantiene la tabella dei file aperti associati ad ogni processo
 - i file aperti sono referenziati da numeri naturali detti *file descriptor*
- le shell *nix quando creano nuovi processi aprono di default 3 file, identificati da 3 costanti definite in <unistd.h>
 - standard input: STDIN_FILENO (solitamente 0)
 - standard output: STDOUT_FILENO (solitamente 1)
 - standard error: STDERR_FILENO (solitamente 2)

open (1/2)

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
int open(const char *pathname, int flags /*, mode_t mode */);
```

- utilizzando la syscall open si richiede al kernel di aprire un file assegnandogli un file descriptor
 - pathname: path del file da aprire
 - flags: OR di costanti definite in <fcntl.h>
 - mode: permessi da assegnare al file nel caso venga creato
 - ritorna il file descriptor assegnato dal kernel, o -1 in caso di errori

open (2/2)

- alcuni flag da `<fnctl.h>`
 - necessari:
 - `O_RDONLY/O_WRONLY/O_RDWR` (mutualmente esclusivi),
accesso in sola lettura / sola scrittura / entrambi
 - opzionali:
 - `O_APPEND` scritte alla fine del file
 - `O_CREAT` crea il file se non esiste
 - `O_EXCL` fallisce se `O_CREAT` è specificato ed il file
esiste (*poor man's locking*)
 - `O_TRUNC` se il file esiste e viene aperto in scrittura azzera
la sua lunghezza
 - `O_SYNC` attende il flush dei buffer dopo ogni scrittura

close

```
#include <unistd.h>
```

```
int close(int fd);
```

- chiude un file
 - fd: file descriptor
 - ritorna 0 in caso di successo, -1 altrimenti
- la chiusura di un file implica:
 - liberazione del file descriptor
 - rilascio di tutti i record lock acquisiti
 - flush dei buffer

lseek (1/2)

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
off_t lseek(int fildes, off_t offset, int whence);
```

- ad ogni file aperto è associata la posizione corrente (*current file offset*): un numero naturale di byte dall'inizio del file
 - operazioni di lettura/scrittura sono effettuate a partire dalla posizione corrente e la incrementano del numero di byte letti/scritti
- lseek permette di cambiare la posizione corrente
 - fildes: file descriptor
 - ritorna la nuova posizione corrente, o -1 in caso di errori

lseek (2/2)

- offset è un intero, la sua semantica dipende da whence
- whence è uno dei seguenti:
 - `SEEK_SET` la nuova posizione corrisponde ad offset
 - `SEEK_CUR` la nuova posizione corrisponde a posizione corrente + offset
 - `SEEK_END` la nuova posizione corrisponde a dimensione del file + offset
- osservazioni:
 - dimensione del file: `fseek(fd, 0, SEEK_END)`
 - file sparsi: `fseek(fd, 1000000, SEEK_END)`
 - seek capability: non tutti i file permettono seek (e.g. pipe, socket, fifo), possiamo verificarlo con `fseek(fd, 0, SEEK_END)`

read

```
#include <unistd.h>
```

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

- read permette di leggere dati da un file aperto, a partire dalla posizione corrente, muovendosi verso la fine del file
 - fd: file descriptor
 - buf: buffer di destinazione per i dati letti
 - per evitare segfault deve avere dimensione \geq count
 - count: numero (massimo) di byte da leggere
 - ritorna il numero di byte letti, o -1 in caso di errori
 - read può ritornare un numero di byte inferiori a quelli richiesti
 - e.g.: fine del file, line-buffering dei terminali, buffering di rete, record-oriented device

write

```
#include <unistd.h>
```

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

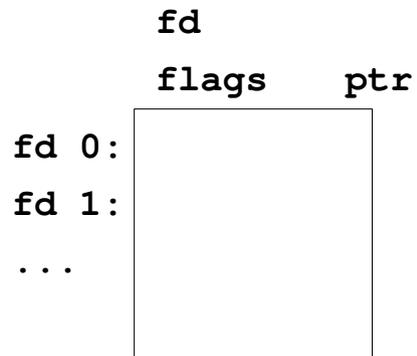
- duale a read: permette di scrivere dati su un file aperto, a partire dalla posizione corrente, scrivendo verso la fine del file
 - fd: file descriptor
 - buf: buffer di origine per i dati da scrivere
 - count: numero (massimo) di byte da scrivere
 - ritorna il numero di byte scritti, o -1 in caso di errori
 - analogamente a read può ritornare un numero di byte inferiore a quelli richiesti

File sharing (1/4)

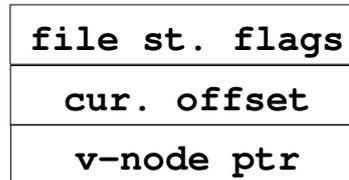
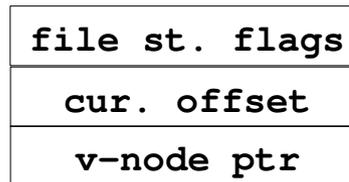
- i file descriptor visti dal kernel
 - ogni processo ha una entry nella *process table*
 - uno dei campi della process table è il vettore dei file aperti
 - ad ogni file descriptor sono associati:
 - i flag del file descriptor
 - un puntatore ad una entry della *file table*
 - ogni entry nella file table contiene:
 - i flag associati al file (come da open)
 - il *current offset*
 - un puntatore ad una entry della *v-node table*
 - ogni v-node entry contiene:
 - le informazioni contenute nell'i-node del file
 - puntatori a funzioni associate al file

File sharing (2/4)

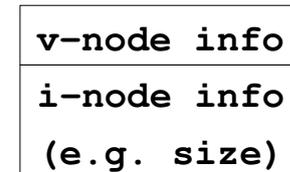
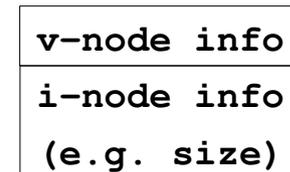
process table entry



file table entry

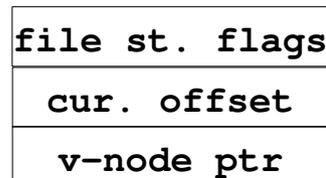
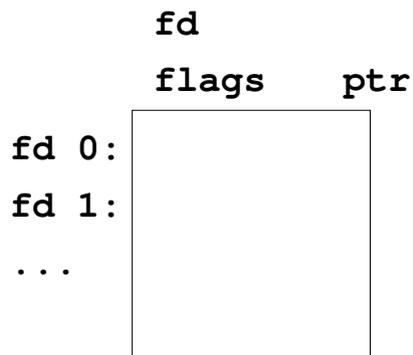
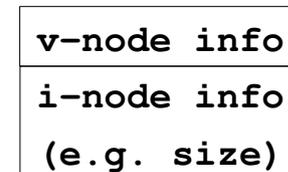
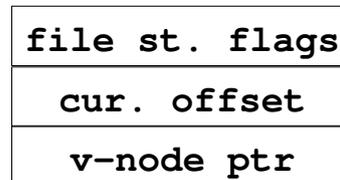
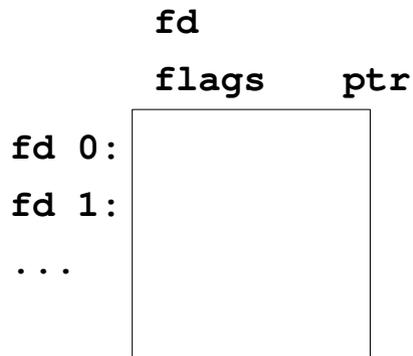


v-node table entry



File sharing (3/4)

- sharing di un file da parte di due processi
 - ogni processo ha il suo offset
 - la dimensione del file è condivisa



File sharing (4/4)

- sharing di un file da parte di due processi
 - *semantica UNIX*:
 - dopo ogni write il processo che l'ha effettuata incrementa il proprio offset; se il file è aumentato di dimensione, il campo size dell'i-node viene incrementato
 - se O_APPEND è stato specificato, prima di ogni write l'offset viene impostato alla attuale dimensione del file
 - lseek modifica solo l'offset del processo che la invoca
 - lseek SEEK_END posizione il file in base alla dimensione del file nel momento in cui viene invocata
 - esistono due modi per condividere una entry della file table tra processi diversi:
 - O_APPEND (come sopra)
 - fork + le system call dup/dup2

dup (1/2)

```
#include <unistd.h>
```

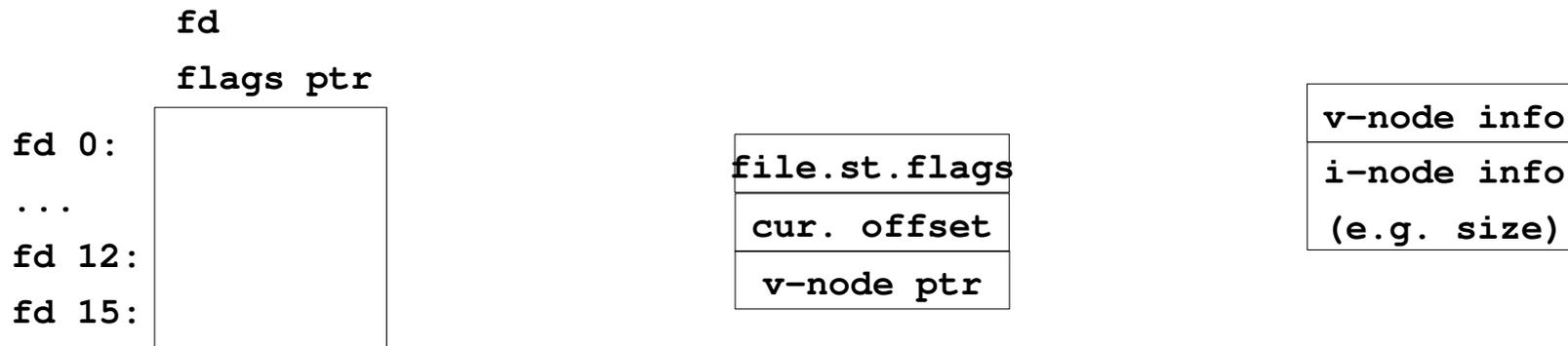
```
int dup(int oldfd);
```

```
int dup2(int oldfd, int newfd);
```

- dup e dup2 duplicano un file descriptor
 - dup sceglie il più piccolo file descriptor disponibile, dup2 permette di specificarlo
 - oldfd: file descriptor che si vuole duplicare
 - ritorna il nuovo file descriptor, o -1 in caso di errori
- invocando fork dopo la duplicazione di un file descriptor è possibile condividere entry della file table tra processi

dup (2/2)

- dopo l'invocazione di dup



- dopo l'invocazione di dup + fork (+ close appropriate)



fcntl

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
int fcntl(int fd, int cmd /* , long arg */);
```

- fcntl permette di modificare le proprietà di un file aperto
 - proprietà:
 - 1.flag associati al file descriptor
 - 2.flag associati al file status
 - 3.processo gestore dell'I/O asincrono
 - 4.record locking
 - 5.(duplicazione di file descriptor)
 - fd: file descriptor
 - cmd: comando da eseguire
 - il valore di ritorno dipende da cmd, -1 in caso di errori

Miscellanea

- altre system call correlate alla gestione dei file
 - stat ritorna informazioni relative ad un file
 - umask modifica la maschera di bit per la creazione di file
 - chmod modifica i permessi associati ad un file
 - chown modifica l'owner di un file
 - link crea link (fisici)
 - unlink rimuove un file
 - symlink crea link simbolici
 - readlink legge la destinazione di un link simbolico
 - utime modifica i tempi di accesso e modifica di un file
 - mkdir/rmdir/opendir/readdir/rewinddir/closedir/chdir
 gestione delle directory
 - sync flush dei buffer del kernel

Libreria I/O standard

- ANSI C definisce una libreria standard di I/O, i cui prototipi sono descritti in `<stdio.h>`
 - la libreria standard:
 - offre un layer d'astrazione sul sistema operativo, può essere implementata anche in sistemi operativi non *nix
 - offre inoltre primitive di più alto livello, e.g.:
 - lettura/scrittura linea per linea
 - gestione automaticamente del buffering
 - formattazione di input/output
 - le syscall di I/O
 - sono implementata solo su sistemi *nix
 - sono spesso più efficienti della libreria standard che deve effettuare copie tra i suoi buffer e quelli del kernel

Riferimenti

- APUE
 - file I/O: cap. 3
 - file e directory: cap. 4