(Stevens)
Capitolo 3: File I/O
(II)

I/O di base

- dup e dup2

dup e dup2

```
#include <unistd.h>

int dup( int filedes );

int dup2( int filedes, int filedes2 );
```

- Descrizione: assegnano un altro fd ad un file che già ne possedeva uno, cioè filedes

Restituiscono entrambe: il nuovo fd se OK
-1 altrimenti

dup e dup2 (II)

- In particolare:

int dup(int filedes);

- restituisce il più piccolo fd disponibile

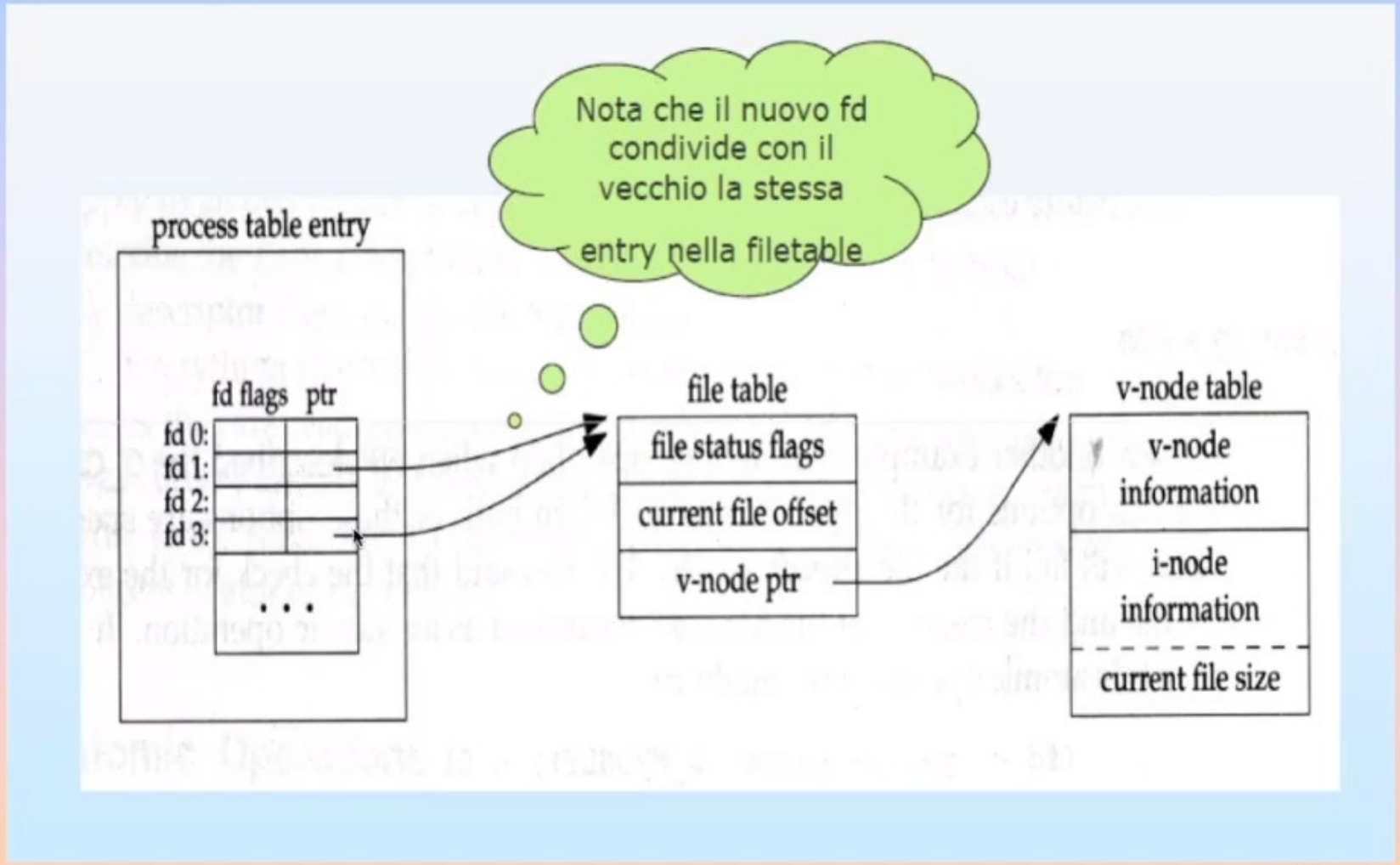
int dup2(int filedes, int filedes2);

- Con dup2 specifichiamo il valore del nuovo file descriptor.
- dup2 assegna al file avente file descriptor ***filedes*** anche il file descriptor ***filedes2***
- Se ***filedes2*** è già aperto verrà prima chiuso.
- Se ***filedes2=filedes*** viene restituito direttamente ***filedes2*** senza chiuderlo
- dup2 è una operazione atomica

dup e dup2 (III)

- Una volta chiamara dup (o dup2), si ottengono quindi due file descriptor che si riferiscono allo stesso puntatore di file.
- Giacché il puntatore a file è condiviso vi è una unica utilità nell'avere un secondo file descriptor: il numero.
- Il nuovo file descriptor sarà il più piccolo disponibile (dup) o indicato direttamente da dup2.
- Questo potrebbe diventare ad esempio utile nel caso di interprocess communication
  - si potrebbe forzare un processo ad avere per esempio il suo standard input e standard output in una pipe,

Cosa succede dopo dup2



(Stevens)
Capitolo 4: Files and Directories

Files e Directories

- stat, fstat e lstat
- opendir e closedir
- mkdir
- rmdir
- chdir
- getcwd

Tipi di files

- In Unix vi sono diversi tipi di files:
  - Files regolari
    - ✓ Il tipo di file più comune, contiene dati. Il kernel di Unix non fa differenza tra testo e dati in forma binaria, ogni interpretazione è lasciata al programma utente.
  - Directories
    - ✓ Un file che contiene nomi di altri files e puntatori alle informazioni su questi files. Ogni processo che ha il permesso in lettura potrà leggere il contenuto di una directory. Solo il kernel potrà invece scrivere.
  - Characters-Blocks Special Files
    - ✓ Files usati, in genere, per rappresentare dispositivi fisici di I/O.

Tipi di files (II)

- Pipe e FIFO
  - ✓ Tipi di file usati per la comunicazione tra processi
- Sockets
  - ✓ Un tipo di file usato per la comunicazione tra processi su una rete
- Link simbolici
  - ✓ Un tipo di file che punta ad un altro file.

stat, fstat e lstat

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

int stat (const char *pathname, struct stat *buf);

int fstat (int fd, struct stat *buf);

int lstat (const char *pathname, struct stat *buf);
```

- Descrizione: danno informazioni sul file 1° argomento

Restituiscono: 0 se OK
-1 in caso di errore

Funzioni stat, fstat, lstat

- Dato un pathname, ***stat*** fornisce una struttura di informazioni relative al file indicato nel primo argomento.
- Il secondo argomento è il nome della struttura che verrà restituita.
  - La definizione della struttura è spesso system dependent ma conterrà informazioni come il mode, il numero di links, l'User ID ed il Group ID del proprietario, le dimensioni, la data dell'ultimo accesso e dell'ultima modifica, etc.
- ***fstat*** ottiene informazioni su un file che è già aperto tramite il file descriptor ***filedes***.
- *lstat* restituisce informazioni sul link simbolico (e non sul file referenziato dal link simbolico).
- Un grande utilizzatore di tali funzioni è il comando shell

  ls –l

  che fornisce informazioni su un file dato come argomento.

struct stat

```
struct stat {
  mode_t   st_mode;    /* file type & mode (permissions) */
  ino_t    st_ino;     /* i-node number (serial number) */
  dev_t    st_dev;     /* device number (filesystem) */
  dev_t    st_rdev;    /* device number for special files */
  nlink_t  st_nlink;   /* number of links */
  uid_t    st_uid;     /* user ID of owner */
  gid_t    st_gid;     /* group ID of owner */
  off_t    st_size;    /* size in bytes, for regular files */
  time_t   st_atime;   /* time of last access */
  time_t   st_mtime;   /* time of last modification */
  time_t   st_ctime;   /* time of last file status change */
  long     st_blksize; /* best I/O block size */
  long     st_blocks;  /* number of 512-byte blocks allocated */
};
```

Tipi di dati di sistema primitivi definiti in <sys/types>

I/O su directory

- Ogni system call di I/O potrebbe essere utilizzata su una directory,
  - a patto che i contenuti della directory non siano modificati.
- Non tutte le call sarebbero però utili
  - Ad es. ***lseek*** non ha granché senso su una directory
- In generale una directory viene aperta per essere letta.
- Per interpretare il contenuto di una directory esiste uno standard header file <***dirent.h***>
  - che definisce la struttura ***struct dirent*** che descrive una entry nella directory.

opendir, readdir e closedir

```
#include <dirent.h>

DIR *opendir(const char *pathname);
    ritorna NULL se errore

struct dirent *readdir(DIR *dp);
    ritorna NULL se non ci sono piu' elementi

int closedir(DIR *dp);

struct dirent
{
ino_t d_ino; /* inode num*/
char d_name[256] /*filename*/
    }
```

mkdir

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

int mkdir (const char *pathname, mode_t mode);
```

- Descrizione: crea una directory i cui permessi di accesso vengono determinati da mode e dalla mode creation mask del processo

Restituisce: 0 se OK,
-1 in caso di errore

mkdir (II)

- La directory creata avrà come
  - owner ID = l'effective ID del processo
  - group ID = group ID della directory padre
    - ✓ vedi altre caratteristiche con man 2 mkdir
- La directory sarà vuota ad eccezione di . e ..

rmdir

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

int rmdir (const char *pathname);
```

- Descrizione: viene decrementato il numero di link al suo i-node; se è =0 si libera la memoria solo se nessun processo ha quella directory aperta

Restituisce: 0 se OK,

-1 in caso di errore

chdir

```
#include <unistd.h>

int chdir (const char *pathname);
```

- Descrizione: cambiano la cwd del processo chiamante a quella specificata come argomento

Restituisce 0 se OK

-1 in caso di errore

getcwd

#include <unistd.h>

char *getcwd (char *buf, size_t size);

- Descrizione: ottiene in buf il path assoluto della cwd

Restituisce: buf se OK

NULL in caso di errore

Informazioni

- Docente: Prof. Bruno Carpentieri
  - e-mail: bcarpentieri@unisa.it

    (nelle e-mail indicare, come subject, il nome del corso, cioé Sistemi Operativi e firmarsi sempre con nome, cognome e matricola)

- Queste slide sono disponibili su Teams.