SO FIB

Nivell lògic:

```
mknod nom
```

`read`: read_disp

`write`: write_disp

Estructuras Disp (S'emmagatzemen en disc (SF) (separem el nom de INODE)

- `Major`
- `Minor`
- `Nom`
- `Inode`
  - Info admin
    - tipus
    - protecció
    - noms
    - propietaris/grup
    - estadístiques
    - mida
    - **Enllaços a operacions dependents**

Físic:

`Taula inodes (TI) -> (RAM):`

Caché de dispositius en ús (1 entrada per dispositiu físic)

Index	#refs	INODE
0	1	i_tty
1	1	i_disp1

Sistema:

`Taula fitxers oberts (TFA) -> (RAM):` - Vinculacions procés vs dispositiu - Una entrada per vinculació (open)

Index	Mode	#refs	INODE	Punter T Inodes
0	RW	-	3	0
1	RW	0	1	1

Index	Mode	#refs	INODE	Punter T Inodes
3	R	0	1	1

```
* Punter T Inodes -> Accés concurrent
```

Procés:

`Taula de canals (TC)`: - (PCB) -> Dispositius que usa el procés - (RAM) Una entrada per vinculació

Index | PT a Fitxers Oberts - | - 1 | 0 2 | 0 3 | 1 4 | 2 * PT a Fitxers Oberts -> Accés compartit

Estructura de dades:

READ:

```
ret = read(4,buffer,512);
```

ret:
- `-1`: error - `=0`: No hi ha res per llegir - `>0`: #byter llegits realment

WRITE:

```
ret = write(3,buffer,strlen(buffer));
```

ret:
- `-1`: error - `>=0`: Escrit realment en bytes

TC: Consulta ref TF Oberts

TFA: - Consulta Mode - Actualitza P lec/exc

CLOSE:

```
close(canal) canal = 3;
```

TC: S'elimina sempre

TFA:

Disminueix #refs

```
    refs--;
```

`Si #refs = 0`: Elimino de la TFOberts

`Si passa`: **TF:** Disminuir #refs

`Si #refs = 0`: Elimino de la TI

REDIRECCIONAMENT D'ENTRADA I SORTIDA:

DUP

```
canal_nou = dup(canal);

canal_nou = dup(4);
```

Duplica el canal passat com a paràmetre

A la primera entrada lliure de la TC.

- `-1`: Error
- `>0` Canal nou > - **TC:** Duplica en el primer canal lliure > - **TFA:** Augmento #refs > - **TI:** Res

DUP2

```
canal_nou = dup2(canal, canalnou);

canal_nou = dup2(3,0);
```

De forma "atòmica"

- Tanca 0 (TC a TPO)
- Duplica 3 a 0

```
// Només és igual si el nou és el 0

canal_nou = dup2(vell, nou); //millor
!=
close(nou);
dup(vell);
```

FORK

```
ret = fork();
```

Crea un procés fill.

- El procés fill hereda casi tot el `PCB`.
- `El fill hereda la TC del pare`.
- `TFA`: S'actualitzen els #refs que toquen.

EXEC

```
ret = exec("path", arg0, arg1, ...);
```

Canvia el binari a executar pel procés. >

- La `TC` es manté.

Pipes:

Anònimes

Són dispositius de comunicació entre processos

Només comuniquen processos emparentats `(pare-fill)` per herència `TC`

- Buffer circular FIFO en RAM
- `Pare` escriu
- `Fill` llegeix

```
int fd_vector[2];
```

```
ret = pipe(fd_vector);
```

`fd_vector[0] // LECTURA si = 3`

`fd_vector[1] // ESCRIPTURA si = 4`

`2 canals`: 2 entrades a fitxers oberts

TC

Index	PT a Fitxers Oberts
3	1
4	2

T FOberts

Index	Mode	P	L/E	#refs	PT a TI
1	R	-	1	1	
2	W	-	1	1	

T FOberts

Index	#refs	INODE
1	2	i_virtual pipe

READ PIPE

```
read(pipe) // buit el contingut de la pipe
```

No és bloqueant (en general)

`-1`: Error

`Si hi ha dades`: Me'n torna tantes com sigui possible (segons el 3r parametre de read)

`Si NO hi ha dades:`

`Si hi ha escriptors` (algun procés amb pipe[1] `OBERT`) -> BLOQUEIG - Fins que hi hagi dades - `pipe[1]` tancat totalment

`Si NO hi ha escriptors = read` -> 0

WRITE PIPE

```
write(pipe) // Afegeix contingut
```

No és bloquejant ~~(en general)~~

`-1`: Errors de "sistema".

`Si hi ha espai`: Escric (retorna segons hagi escrit, limitat pel 3r paràmetre).

`Si NO hi ha espai`: BLOQUEIG (hi han lectors) fins que hi hagi espai.

`Si NO hi ha lectors`: Rebo SIGPIPE - Acció per defecte: ACABAR - Si SIGPIPE està capturada

```
write = -1

errno = EPIPE
```

`BIDIRECCIONAL` (TECNOLÒGICAMENT)

`FUNCIONALMENT` no ho és

Utilitzem `2 pipes` per fer-ho

Fill

El fill es buida

Read es bloquejant

Hi ha un escriptor -> El propi Fill

Tancar sempre els canals que no utilitzem (Sobretot PIPES)

Pipes amb nom en el SF

```
mknod p nom
```

```
// es bloquejant fins que s'obri l'altre extrem
```

```
open(nom, R)
open(nom, W)
```

open:

Obrir fitxer

```
ret = open("ruta/nombre",   mode,       flags);

                            O_RDONLY    0
                            O_WRONLY
                            O_RDWR
```

Returns:

`ret = -1` Error

`ret = canal`

Crear un fitxer:

```
ret = open("ruta/nom", {O_WRONLY | O_CREAT | O_TRUNC, Permisos)
                        O_RDONLY}                     Si existeix
                                                      es sobrescriu
```

Preguntar existencia del fitxers:

```
ret = open("ruta/nom", MODE | O_CREAT | o_EXECL, Permisos)
```

Si existeix (obert):

`ret = -1`

`errno = EEXIST` (File exists)

lseek:

Unix: A nivell de SO

Els fitxers són sentències de bytes

P lec/esc

Accés directe per posició

`lseek`: cambia el P lec/escrit

```
ret = lseek(canal,  desplaçament en bytes,  Des d'on)

                    (qualsevol valor +/-)   SEEK_SET
                                            SEEK_CURL
                                            SEEK_END
```

Des d'on:

SEEK_SET: des de l'inici del fitxer

SEEK_CURL: des de la posició actual de P lec/escrip

SEEK_END: des del final del fitxer (EOF)

Returns:

`-1`: Error

P l/e < 0

P l/e > EOF **NO ERROR** -> Entre EOF i P l/e hi haurà basura

`ret >= 0`: Posició del P lec/escrip desd de l'inici del fitxer

Sistema de fitxers:

Virtual File System:

`Disc inmanejable`

Accés a dades compartimentats (nom)

Organització -> Directori (crea la jerarquia)

Directori:

Fitxer especial (no accessible mitjançant syscalls)

Lliga **noms** amb el **contingut**

```
ls -l
```

tipus permisos nº links usuari grup … noms
d
-
p
s
c
nom **inode**
. Referència a ell mateix
. . Referència a directori pare
fitxer.txt 7 ->

Generen jerarquia (arbre invers)

```
. / . .
```

Punt d'entrada

Arrel /

/

. Inode de #0 /

. . Inode de #0 /

Accés absolut

Comença per /

```
/__ / __ / __ .txt
```

Accés relatiu

a partir de `CWD` (Current Working Directory)

Mai comença per `/`

```
__ .txt
```

Directoris en graf:

`Arbre de directoris`

+noms: Graf cíclic

- **Hardlinks:** Accés al contingut`* Hardlink només permet al mateix sistema de fitxers`
- **Softlinks:** Accés és a un altre nom (Accés directe)

nom	inode	tipus
.	-	
. .	-	
A	7	Hardlink
B	7	Hardlink
C	25	Softlink (El contingut de C és un altre noms) `Estàn marcats: Així podem obrir el destí del accés directe`

C (Accés directe)

|

25

|

A/C/D (Contingut)

Descriptor fitxer: INODE

- Tipus -> `c / b / p / l (soft) / -`
- Mida
- nº enllaços -> **nº de hard links**
- Atributs (rwx)
- Accés a ops dependents

Discs organitzats en particions:

Sector:

Unitat de transferència del fabricant (512, 2 kb...)

Bloc:

Unitat de transferència del SO

La mida és múltiple

```
Mida(bloc) = num*Mida(Sector)
```

Mida(bloc) comú pel sistema

VFS

Els discos/particions es monten en punts a partir de la `/` arrel global

/root/

part1

ext1

/home/

part2

ext4

Contingut de les particions linux (ext2,3,4):

- BOOT (si és partició d'arrencada)
- SUPERBLOC
  - Contingut en metadades del SF
    - Formatejat
    - Gestió d'espai de la partició
      - Llista inodes lliures/ocupats
      - Llista Blocs de dades lliure/ocupats
    - Mida
    - Referència inode

Secció de inodes:

Llista dels inodes reservats a la partició (ocupats o no) - Tots els inodes del SF (`/ es el inode 0`)

Secció de dades

BLOCS amb contingut dels fitxers

METADADES (SUPERBLOC)

DESCRIPTORS INODES (disp, directori, fitxer)

DADES

- Contingut dels fitxers
- Contingut dels directoris (espai de noms -> jerarquia)

Bloc de dades directori:

-	-	-
.	0	Apunten al inode d'aquell fitxer
. .	0	
A	1	

B | 2

Assignació d'espai a dades:

Contigu:

F1 F2 F3 F4

EXEMPLE: CD's, DVD's... Dispositius de només lectura (ISO9660 - Joliet)

No contigu:

FAT (File Access Table)

FAT /BUIT/ /BUIT/ FAT	
nº bloc	**on continua**
0	* (adreça 3, per exemple)
1	FREE
2	FREE
3	EOF

DIR: Nom + bloc d'inici

Esquema multinivell: Linux (UNIX):

NOM --- INODE (a la llista d'inodes)

```
Mida inode = Mida Bloc (4kb)
```

CONTINGUT DE L'INODE - Mida - Tipus - Permisos - Op dependents - **ENLLAÇOS** a blocs de DADES - sr fitxer - normal - directori

10 enllaços directes a dades `És rapidissim` --- `-> dada`

1 enllaç indirecte `2 accessos` --- `-> -> dada`

1 enllaç indirecte doble `3 accessos` --- `-> -> -> dada`

1 enllaç indirecte triple `4 accessos` --- `-> -> -> -> dada`

TFA -> Mode + P l/e

T INODES -> Caché de llista d'inodes del disc (Inodes en us)

BUFFER-CACHÉ -> Emmagatzema blocs

```
- INODES
- Blocs de dades
```

RESUM

SF

Jerarquia: directoris

Graf:

NOM

Hard links

Soft Links

Boot Super bloc indoes Blocs de dades

Fitxers / directoris

Arrel directori -> / (inode 0)

```
Directori: 1 INODE + 1 (al menys) BLOC DE DADES
```

Nom INODES

.	Inodes
. .	Inode
A	Inode

DISPOSITIU -> 1 Inode

```
Fitxers "normals": 1 INODE + 0 o més BLOCS DE DADES
```

```
Estruct Inode:
```

- Info admin
  - Mida
  - Data de Creació
  - Propietari
- Enllaços a crides DEP
- Si normal o directoris
  - Enllaç a bloc de dades

HARD LINK VS. SOFT Links

HL: 2 noms apuntant al mateix inode (dues entrades al disc dur)

SL: Fitxert marcat com a "link" (a l'INODE) i el seu contingut (BDades) es la ruta a l'altre fitxer

Disc SF

Super bloc INODES Blocs de dades

Procés TC

- `Sistema operatiu`
  - TFA (open/close)

- T.Inodes: Caché INODES de fitxers en ús
- Buffer-Caché: Caché de blocs (Inodes + blocs)

OPEN

```
open(ruta)
/home/alumne/s7/a.c
```

- Navegació
  - Inode /(0) + BLOC /
- Inode 8 + BLOC /home

`open:` acaba al portar a memòria el inode de "a.c" - En la TI (si no acaba) - `NO carrega blocs de dades` (o + refs) - +1 entrada a la TFA (mode segons els paràmetres) P l/e = 0 - +1 entrada a TC

Super bloc INODES Blocs de dades

1. Crear /home/alumne/b.c > GRAVAR-LOS: > - Si no existeix: Demanar inode al SUPERBLOC > - TI | > - Modificar bloc "alumne" > - Afegir "bc" nom+I > - Modificar INODE alumne (mida)
2. Actualitzar Super bloc en el disc

READ

```
read(_,_,long)
```

- `long`
  - transformar a blocs
  - +P l/e (TFA)

Augmenta el P l/e en ret

lseek

- No afecta el disc
- Canvia el P l/e en TFA