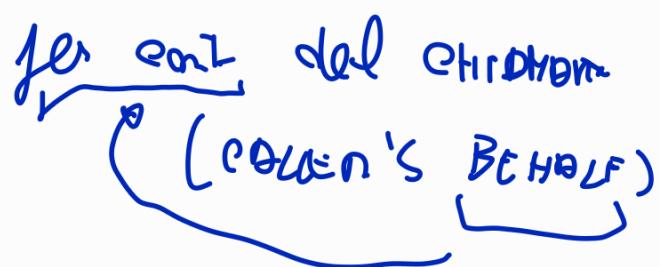


## BOOK

System call → richieste al kernel di eseguire determinata funz.



SYSCALL → elabora la system call

## ESECUZIONE DELLA SYSTEM CALL

WRAPPED FUNCTION → CHIAMATA DI SISTEMA

1 il program chiama la wrapped function in C

2 la wrapped inserisce nei registri formalmente gli arguments SYSCALL

H syscall → piccola libreria che invia la system call  
scritta in assembly e ha il nome sys.  
di ARGOGENI

3 la wrapped mette i return system nel registro

4 fa TSS → è la lista dei programmi in memoria

TRIDP: consente di leggere o scrivere memoria dell'utente  
di sistema

interruzione di segnale

invocare una routine  
del kernel

la priorità 0-15

(WEEKEND weeks)

TRDP:

informazioni riguardanti quale CPU gestisce un programma o quale utente cerca di eseguire un certa privilegio

la CPU trasferisce il controllo al kernel

↳ fa wrapped function → chiamare la **try** → exit

la CPU trasferisce il controllo al kernel

Per fare 1 wrapped function → 2 algorithmi SYSCALL nei RISIST

→ 3 numeri SYSCALL nei registi

→ 4 try per kernel mode

5) kernel esegue la SYSCALL

1) valutazione del numero SYSCALL

↳ controllo dei privilegi

2) make all numbers division in un register

3) return in 05th - 06th

6) Estimate Number or Return

1) WRAPPER FUNCTION

2) Placeholder returning TO register

3) SYS CALL NUMBER TO DECIMAL

4) TRAP TO EXCEPT MONG

5) VALIDATE SYSTEM NUMBER

(CHECKS PROPERTIES)

RETURN A VALUE IN A REGISTER

RETURN USER NUMBER

6) Checks SYS CALL RETURN VALUE

LIBRARY FUNCTION

→ take different standard C

→ OBGETS: I/O, FILE, PATH, STRING

make function user SYSTEM CALL & make no

## ERROR HANDLING

Molti chiamati di sistema e funzioni librerie  
+ norme per gestire le eccezioni / fallimenti

#1 mette "-1" per indicare errori

→ un chiamato di sistema non modifica mai ERRNO  
↳ esempio -1 è un fine  
↳ quando fallisce → il suo ERRNO viene impostato

In un ambiente MULTITHREADING ENVIRONMENT, ogni thread  
ha il suoERRNO ID

ERRNO	1	2	3
OPERATION	NO	NO	NO
NON	FILE OR		PROCESS
PERMISSIONS	DIR		
EPERM	ENOENT		ESRCH

altri: EINTR → chiamata di sistema interrotta da un  
segnale?

EBADF → file DESCRIPTION non valido

# PATH SYSTEMS

Tanti diff. per le info di SISTEMA → PROCESS ID, USER ID, FILE OFFSET

USER ID

FILE OFFSET

Utilizzano i dati matrici di C ma non è finalabile.

→ se la lunghezza in un intero è 4, in un altro 17 sarebbe ancora 8

→ anche nella stessa struttura possono essere diversi

POSIX ha definito un tipo diff.

→ in modo che ogni struttura abbia il suo

es. TYPEDEF INT PID\_T => PID\_T MYPID;

L'SISTEMA

DBTB

POIX  
REQUIREMENT

DESCRIZIONE



Attenzione A1: DESSI POSSONO A PRINT

↳ FILE → getpid() → PUÒ ESSERE CORIO  
Ma anche LUNGO  
↳ Eccolo IL LUNGO  
e SI PROSPANO  
BUTT

→ aggiungere LIMIT quanto viene chiesto dal SISTEMA

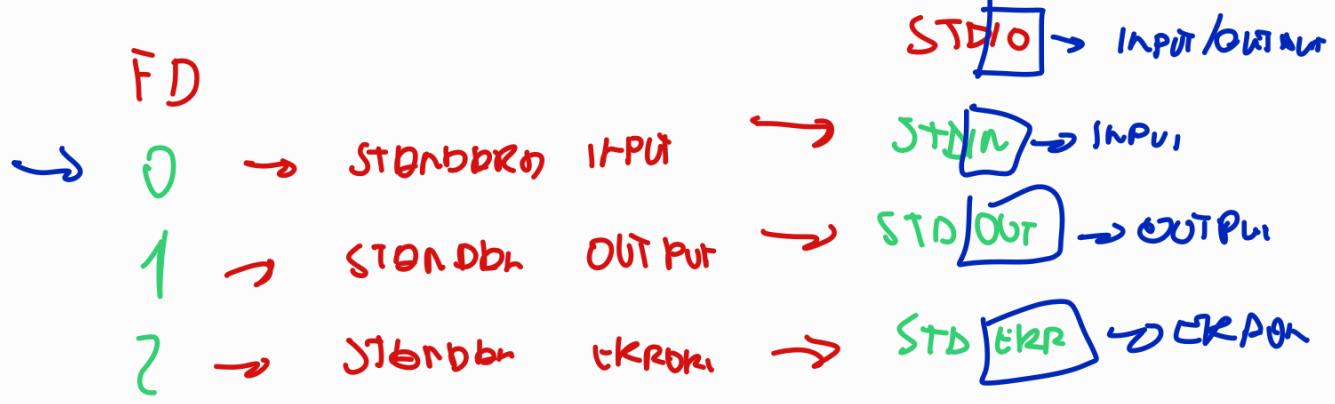
ff+ → come lungo %zu e %zd for  
SIZE\_T & SSIZE\_T

STDIO PACKAGE FOR FILE I/O

SYSTEM CALL → OPEN, CLOSE, LSEEK, READ, WRITE

OPERAZIONI I/O → FDS (FILE DESCRIPTION)

↳ numeri interi non memorizzati  
↳ 30 qualcosa → file, pipe, FIFO, ecc



## System call FDs

`open()` → ritorna il file descriptor

`read()` → input

`write()` → output

`close()` → chiude il file descriptor

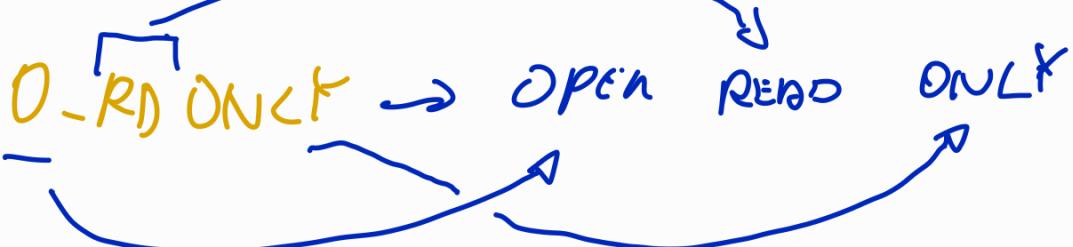
## Open()

→ apre il file se esiste oppure lo crea

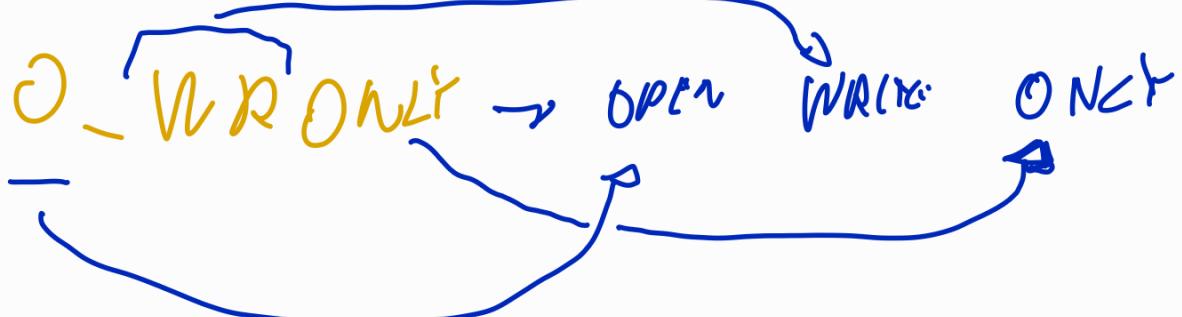
→ ritorna un FD (INTERO NON NEGATIVO)

- parametri
  - 1) PATHNAME → identifica il file da aprire
  - 2) FLAG di controllo: O\_RDONLY / O\_CREAT
  - 3) MODE → attributi

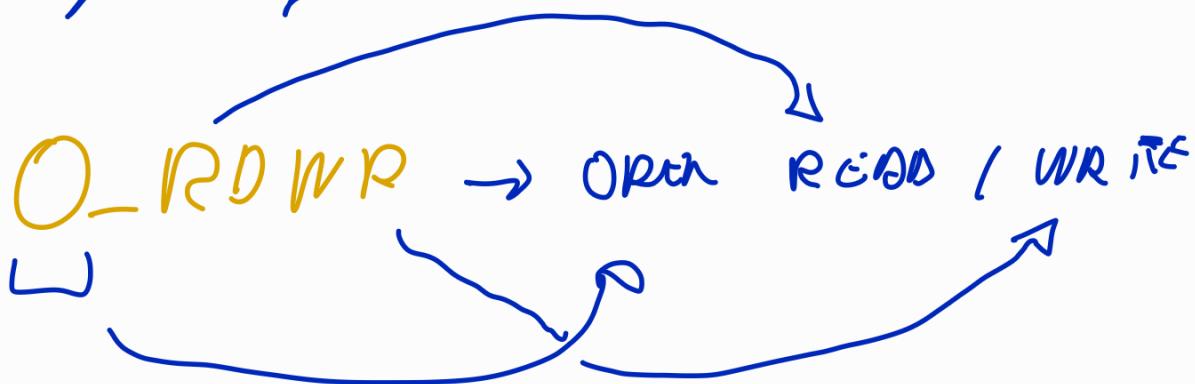
P2b6



sys il file in LETTERA



sys il file in SCRITTURA



FLAG ULTERIORI

O\_CREAT <sup>FILE CREATION</sup> → sys il file se non esiste → se non esiste, sys solo se file esiste

O\_APPEND <sup>FILE STATUS</sup> → scrivere in ultimo file

O\_EXCL <sup>FILE PROTECTION</sup> → ore esclusività → il file non deve esistere  
↳ non si può com O\_CREAT  
↳ non si può scrivere in file già esiste

PERMESSI

S\_IRUSR → READ PERMISSION, OWNER  
↳ 0400 → il proprietario del file  
ha il permesso di leggere

S\_IWUSR → WRITE PERMISSION, OWNER  
↳ 0200 → il proprietario del file ha il permesso  
di scrivere

### ESEMPIO APERTURA FILE

1) `int fd = open ("test.txt", O_RDONLY)`  
↳ OPEN READ ONLY  
↳ APERTURA FILE IN LETTURA

2) `int fd = open ("myfile.txt", O_RDWR | O_CREAT | O_EXCL  
S_IRUSR | S_IWUSR);`

- 1 → file aperto in lettura
- 2 → file viene creato se non esiste già
- 3 → se il file già esiste da errore
- 4 → file con permessi di lettura solo all'utente PROPRIETARIO
- 5 → file aperto in scrittura solo per l'utente PROPRIETARIO

3) ~~O\_TRUNC~~ → elmin id content (BFR)

↳ deve avviare file scrivo

fd : gte ("app.log", O\_WRONLY | O\_CREAT | O\_TRUNC

| O\_APPEND) S\_IRUSR | S\_IWUSR);

O\_WRONLY → apre id file in scrittura

O\_CREAT → crea id file se non esiste già

O\_TRUNC → elmin tutti i BFR del file

O\_APPEND → aggiunge content alla fine del file

-  
S\_IRUSR → permessi di lettura solo per il proprietario del file

S\_IWUSR → permessi di scrittura solo per il proprietario del file

READ() →

SIZE\_T

↳ SCOSTATO → FILE POSITION E  
NECESSARIO

read (int FD, VOID \*BUFFER, SIZE\_T COUNT)

↓

fronte di

numero di BFR letto

descrittore

diffuso

→  
non termina

Par. NUL  
Byte

VALORI DI RITORNO

{  
+1 → numero di byte letti  
0 → fine del file (EOF)  
-1 → errore

WRITE() → mette dati in SSISTE

WRITE( int FD, const void \*BUFFER, size\_t COUNT )

↓  
FILE  
Descriptor

↓  
indirizzo  
di dati da  
scrivere

↓  
numero di  
Byte

VALORI DI RITORNO → {  
numero di Byte  
-1 → errore

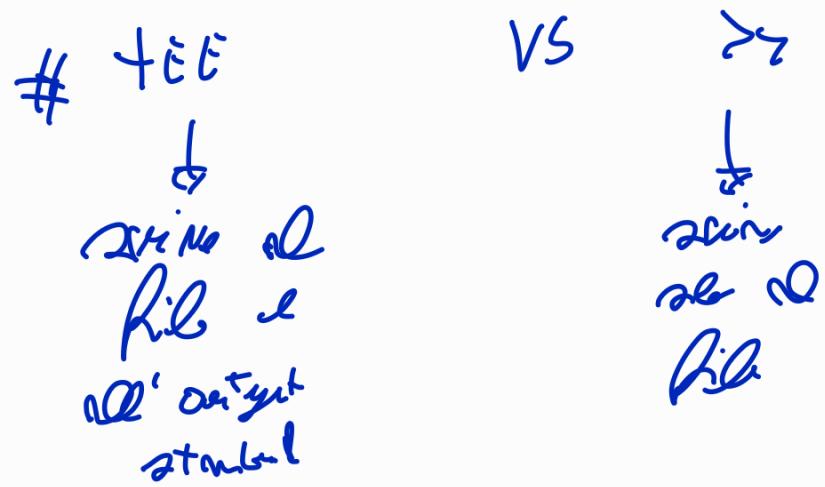
CLOSE()

int close(fd) RITORNO 0 se fu SUCCESSO  
-1 → errore

→ si può controllare il suo contenuto

↳ si è controllato che due volte la stessa file

## ESERCIZIO

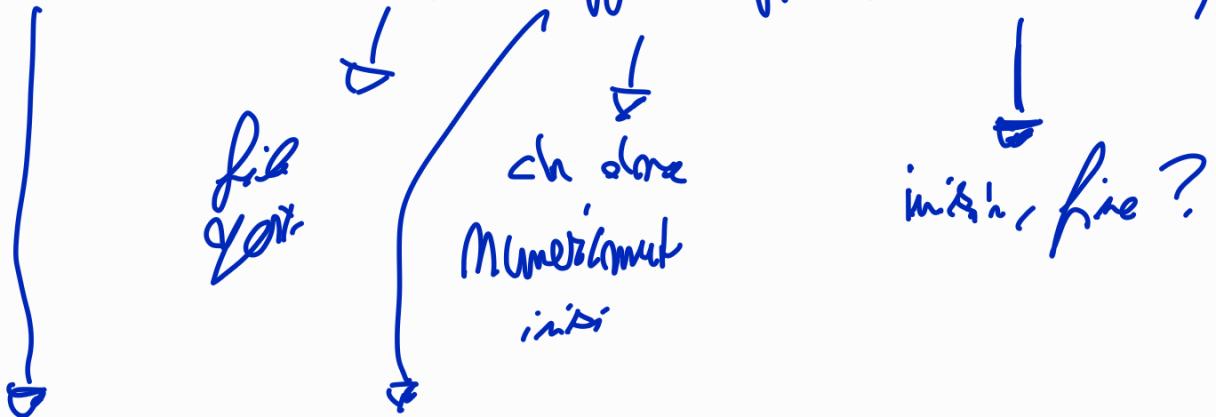


## FILE OFFSET

↳ setta lo byte 0 → apre il file

↳ **OFF-T** → byte 0 → apre il file

SEEK → (int fd, off\_t offset, int whence)



$\text{off\_T} \rightarrow$  ritorna l'off+ del file

TOD INIZIO

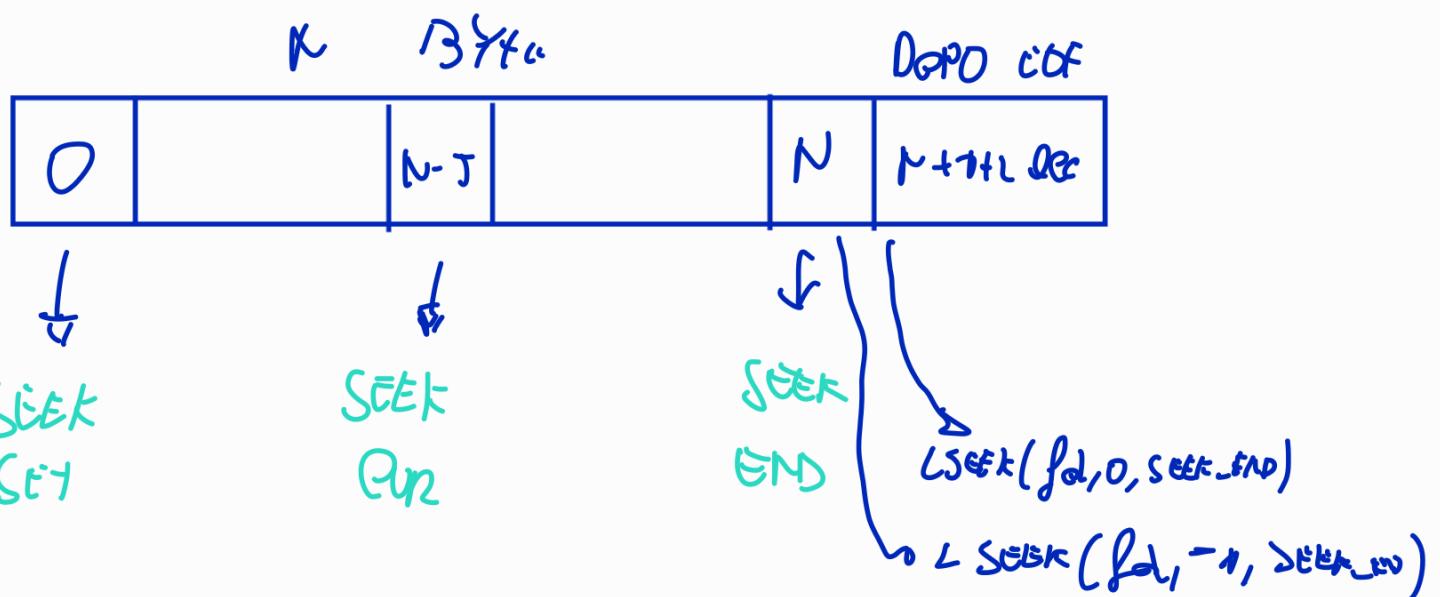
altrimenti file man mano SEEKABLE  $\rightarrow$  RICERCHABILI

Offset  $\rightarrow$  Byte position

{

- SEEK\_SET  $\rightarrow$  Start File
- SEEK\_CUR  $\rightarrow$  Posizione corrente
- SEEK\_END  $\rightarrow$  Posizione finale (EOF)

$\rightarrow$  file di N byte



Esempio

Offset + LSEEK(fd, 0, SEEK\_SET)

↳ INTRO FILE

OFF + LSEEK (fd, 1000, SEEK\_SET)

↳ BYTES 1000

OFF + LSEEK (fd, 0, SEEK\_END)

↳ PREVIOUS BYTES NOT EOF

OFF + LSEEK (fd, -1, SEEK\_END)

↳ within BYTES of file

## RELATIONSHIP BETWEEN FILE DESCRIPTION & OPEN FILE

→ PIG FILE describes current file position like other file

↳ SYSTEM-WIDE TABLE (OPEN FILE)

TABLES ARE FILE OR LINKS OR entries

( fd FD TABLE points ~ questi.)

## FILE TABLE

FILE OFFSET	RDONLY	?
FILE STATUS FILE	P WRONLY	
DIFFERENT FD INODE	N / W / R - W	RDWR

In SYSTEM WIDE TABLE si ricava che file INODE  
nel file system

### I NODE TABLE

{  
file type  
file permissions  
file properties}

che riporta il file descriptor nella stessa tabella OPEN FILE

↪ contiene il file OFFSET

↪ se cambia il file offset da un file

↪ ogni modifica anche dell'altro  
↪ PROBLEMI DI SINCRONIZZAZIONE?  
↪ si applica anche negli STATUS FLAGS

gli status flags → nell'open file table

mentre il FD flag → privato per ogni open e file descriptor

\* KCMO → vedere se due fd sono simili per lo stesso OFD

### DUPPLICATING FILE DESCRIPTORS

int DUP(int oldfd);  
{|  
↪ file descriptor list

Vi ritornano informazioni sul file → METADATA

#include <sys/types.h>  
#include <sys/stat.h>  
#include <sys/conf.h>  
#include <stropts.h>  
#include <stropts.h>

fd[0] → STDIN\_FILENO  
fd[1] → STDOUT\_FILENO  
fd[2] → STDERR\_FILENO

0, 1, 2 → sempre open

close(STDERR) → 2 file

New FD = dup(STDOUT) → USO 2

↳ 2 > 1 ↳  
dup usato

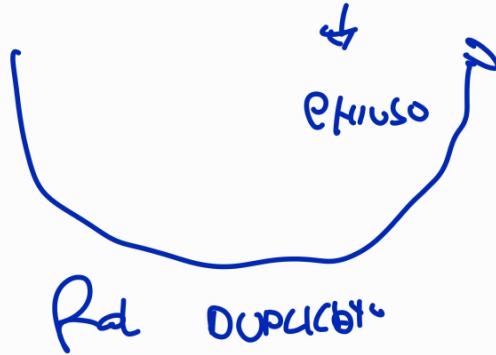
il numero  
più basso

e se si chiude ↑ PRMO?

dup2(OLDFD, NEWFD) → chiude NEWFD

↳ oldfd viene aperto ANCHE SU NEWFD

→ dup2(STDOUT\_FILENO, STDERR\_FILENO);



Start

→ recuperare informazioni sul file → METADATA  
↳ già fatto su statbuf

{ ... } = inoltre PATHNAME-

- S<sub>I</sub>AT() → info sul file
- LSTAT() → informazioni sul LINK SIMBOLICO
- FSTAT() → info sul file fd

## STRUCTURE (ALCUNI)

(FORMATO)

→ S<sub>I</sub>-INO → INODE NUMBER

→ S<sub>I</sub>-SIZE → file SIZE (BYTE)

→ {  
 A-TIME  
 M-TIME  
 C-TIME } →  
 TIME D'MODIFICA  
 DI ACCUNDO  
 COSE

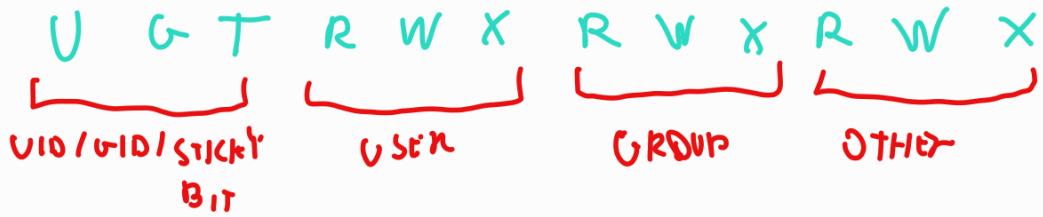
S<sub>I</sub>-UID & S<sub>I</sub>-GID → OWNER ST-IDs (PROPRIETÀ FILE)

↳ USER

IDENTIFICATION → rappresentazione dell'utente  
nel kernel LINUX

DIRETTORE : X : 1000 : 1000  
 VID      GID

→ STATBUF. S<sub>I</sub>-MODE  
 S<sub>I</sub>-MODE → due parti → file TYPE → BITS SUCH AS MODE  
 → file permission → FILE PERMISSION → 9 BITS



**DIRECTORY** ≈ **LINK**

↳ MEMORIZZANO ANNO STESSO MODO DI UN FILE  
(ha controlli già come directory)

→ tabella che associa i nomi  
dei file all'inode

# LS -i  
lo tif vedrem  
il nome del  
file

file max limit bytes ~ 255

DIR	come in Link
FILE NAME → INODE	
F1	1
F2	2
F3	3
F4	4
F5	5
F6	6

→ PIÙ NOMI DI FILE POSSONO ESSERE  
ASSOCIAZIONI ALLO STESSO DIR

**HARD LINK**

→ trawl il file system → i controlli si creano in doppio filo  
↳ in quelli del file originale

LS -L I → INODE

↳ information su PERMESSI, TYPE ecc

→ ogni INODE → HB OR LINK COUNT

↳ RAM decrementa il link counter  
nella INODE

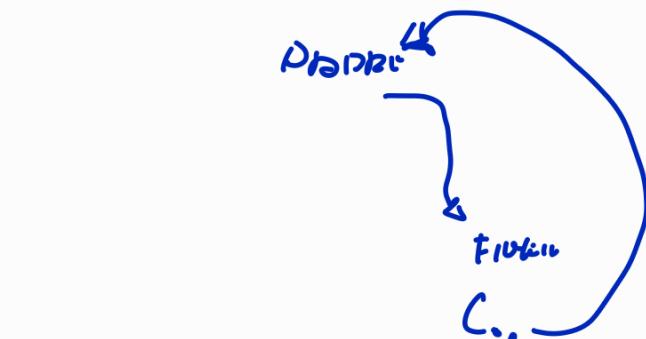
→ NO LINK SU FILE DI ALTRI FILE SYSTEM

↳ ogni file System HD / SVOL / MABT

→ file man create cycle → man si crea link su  
DIRECTORY

↳ Verifica ogni inodo è COSTOSO

.. → DIR PBDM



↳ Se un file

ha TANTI PDDR? (H2 < D  
dynam. Tl01k)

# SYMBOLIK LINK / SYMLINK / SOFTLINK

LN ->

→ INDIR. COMMISSIONE

→ destinazione del collegamento → TARGET

POTREBBE  
ESSERE

ANCHE UN  
ALTRO SYMLINK

HARD VS SOFT

↓  
INDIR.  
NOME

↓  
PATHNAME

NON DIPENDO

DAL LINK COUNT

↓  
ELIMINAZIONE

DI UN SOFT LINK PRODUCE ERRORE

HARD → NON SI COMMETTONO

TUTTO FILE SYSTEM → NON RECOM

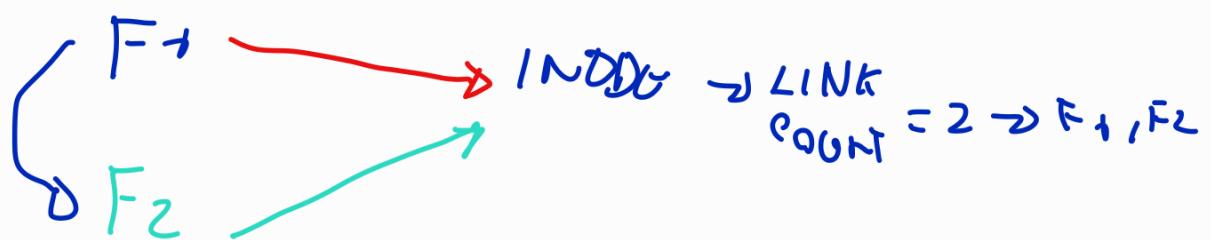
DB EDITOR

CALCI

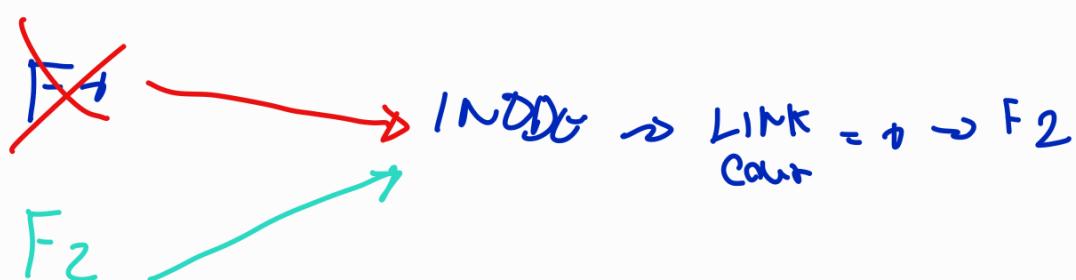
↓  
SOFT SI

PER UN HARD LINK PREGIUDIZIO VALIDI

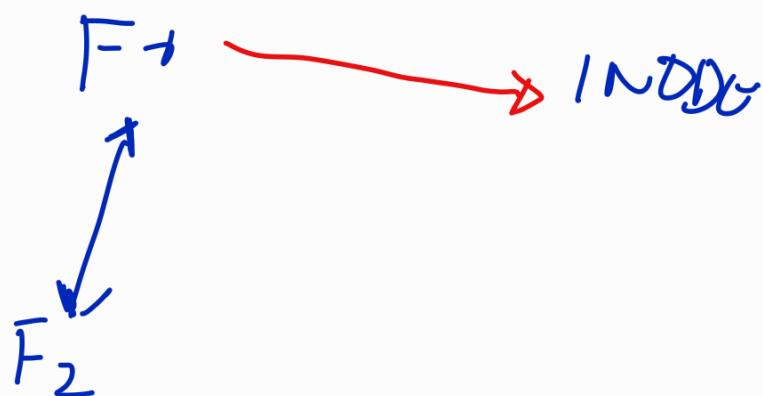
# file nel file system → collegamento da' INDIR.



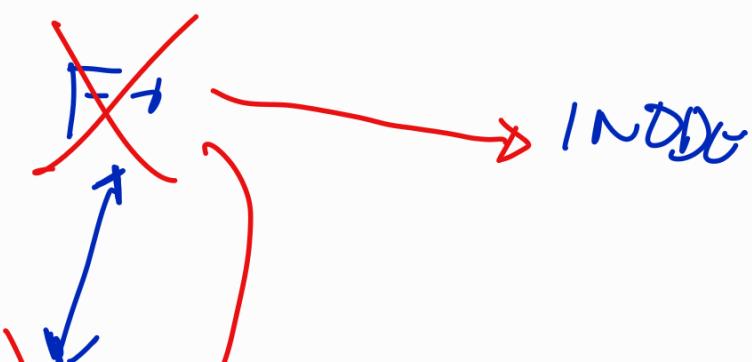
HARD LINK → Open in alt  
file and file system



## SYMBOLIC LINK



↪ SYMBOLIC LINK → Link of  
POINTING TO



~~F~~ A  
FILESYSTEM HARD LINK IN C

int link (OLD PATH, NEW PATH)  
↳ NON OVERWRITE  
↳ SISTER DIR

↳ LN OLD PATH NEW PATH

PER REMOVE → UNLINK (PATH NAME)

↳ -> link counter nel file → ogni volta a 0  
↳ NO A DIRECTORY il file è ELIMINATO

LINK COUNTER → è il kernel che conta i file open (fd) che fanno riferimento al file

UNLINK → se c'è almeno un file è open  
in un altro process

REMOVE DIRECTORY

→ int remove (pathname);

↳ ~~RMDIR~~ ↳ UNLINK FILE ↳ 2 RMDIR ()  
DIR

F1	IN1
F2	IN2
F3	IN3
F4	IN4

UN UN UN UN

altre link → anche SYMLINK (JLDPATH, MCLPATH)

**Symlink()** → può puntare a un file esistente

o un file esistente

↳ LINK  
PERMESSO

### C Scrittura

TARGET = "ORIGINAL.TXT"

LINKPATH = "LINK\_TO\_ORIGINAL.TXT"

↳ SYMLINK (TARGET, LINKPATH)

**Readlink()**

che cosa memorizza il buffer?

p

SS140\_1 READLINK ( PATHNAME, BUFFER, SIZE\_T BUFSIZE )

↳ legge il PERMESSO DEL LINK SIMbolico

↳ Dimensione

### CURRENT WORKING DIRECTORY

↳ Oggi processo che ha programmato  
WORKING DIRECTORY (CWD)

↳ GetCWD()

↳ (char \* cwd, char \* buffer, size\_t size)

↓  
PATHNAME  
OP chro

↳ NUMERO  
DI BYTE VALORI

ChDIR → cambia il CWD

FCHDIR → Cambia il CWD nella dir che è:

... una o più directory

Riflessione su file ...

## SEARCH DIRECTORY

- apriamo → `OPENDIR()`
- cercare dentro → `REaddir()` →
  - FILENAME
  - INODE
- per un intero album
- DI FBRTYLR → `NFWC()`

## PROCESSI

Un processo è un programma in esecuzione.

Più precisamente è l'esecuzione di una **ISTANZA** del programma.

Ogni istanza è iniziale che gira un programma nella memoria, il sistema operativo. Che in esecuzione, il processo è un copy ottimale del programma → risoluzione, memoria, siti.

Ogni processo ha il suo PID, process ID, identificante del PID-T, che identifica il PID, è un GETPID() → ritorna il PID del chiamante.

PID massimo: 32767 (Linux)

Il kernel non si ricorda mai PID, limitandosi numeri già finiti, se tutti le slot del PID sono usate, la funzione fallisce.

La funzione è un sistema per creare processi figli.

Ogni processo non è "unico" ma può avere un processo padre, che identifica il processo padre in un PID-T getppid(void) (P\_PID → PARENT PROCESS ID).

È garantito un determinato "legame" tra processi padre e processi figli.

Il processo padre deve informare della morte del processo figlio.

La creazione di processi forma un albero, alla radice si creerà il processo INIT con PID 1, intitolarci il padre di tutti i processi orfani.

## PROCESS Manager

In maneggi: Nostante si un processo si divide in più segmenti:

1) TEXT: istruzione e funzioni MACCHINA, mentre è solo comune per entrambi MODIFICHE

→ PIÙ PROCESSI POSSONO CONDIVIDERE LO STESSO CODICE IN MEMORIA

2) INITIALIZED DATA: variabili statiche e globali che sono espressamente dichiarate, ovvero dichiarano il file del programma legge; valori quindi che il processo viene creato

3) UNINITIALIZED DATA: variabili statiche e globali che non sono espressamente dichiarate → inizializzate a 0 quando il process viene creato

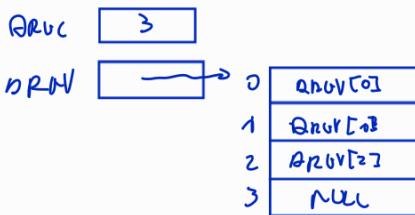
4) STACK: "magnetone" per le variabili locali delle funzioni

5) HEAP: area dove la memoria viene dinamicamente allocata e  
dall'utente viene dealloca e free

## Contenuto della struttura

ARGV → numero di ARGUMENTI

CHAR \*ENV[1] → array di puntatori a stringhe



Ogni processo ha le sue VARIABILI D'AMBIENTI # INHERITS:

↳ in maniera /recursiva eredita una copia dell'ambiente proprio <sup>eredita</sup>

## /PROC FILE SYSTEM

→ è un PSEUDOFILE SYSTEM che dà informazioni sul KERNEL

↳ strutturato come un file-system → file system per le strutture dati del kernel e i processi in esecuzione

↳ come se fosse un file system

→ i file non sono PHYSICAL FILES ma sono FILE DESCRIPTION MONITOR SW

DISCO → logiche unità (DM-TIME-FILE) quale regola modifica

## SEMANTICHE → SOFTWARE INTERRUPTS

↳ verifica al process che un event si è verificato

ASYNCHRONOUS → non si sa se si rispetta  
una scadenza o meno

→ ogni regola ha un suo valore intero UNICO

→ fissa direz. memori. del kernel  
o da un altro processo  $(\text{free}(\text{p1}, \text{s1}))$

### COSTO DI STONDO:

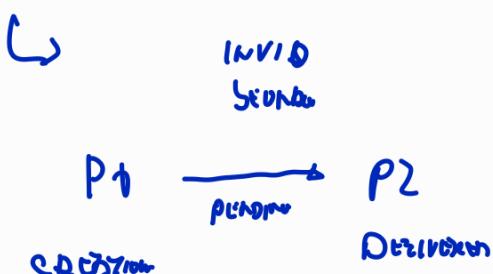
1) desidero un indirizzo di mem. non esist.

↳ SIGSEGV

2) interrup. symbol  $\rightarrow$  SBLINT (termina il thread)

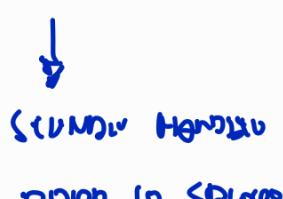
3) gress figli terminati SIGCHLD

→ generazione (ereditazione) e spedizione del segnale



se può liberare l'indiriz. del segnale (SPECIFICO)

aggiungerlo al Symbol Mask  $\rightarrow$  blocca le sezioni



SPECIFICHE DEL SEGNALE

cosa fa un process?

- regola remota sul kernel
- il process termina → "KILLED"
- il process termina e crea un CORE DUMP
  - ↳ core dump → file → contiene l'immagine della memoria del processo al momento della conclusione
- posso usare quel file nel debugger
  - ↳ cogliere le righe nel programma al momento della sua esecuzione
- STOP PROCESS → continua esecuzione

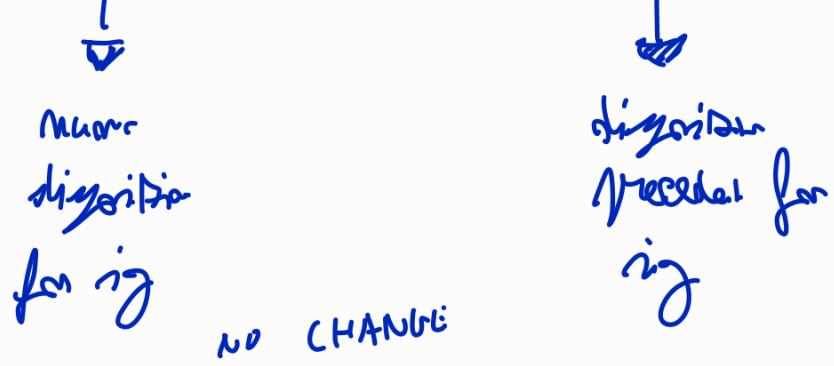
SIGKILL e SIGSTOP non fanno effetti CONTINUITI, BLOCCANTI o GUARDANTI

### "COMBINARE" IL SEGNALE

- COMBINARE il segnale → escludere il secondo → è gentile tranne funz. dell'inter
- ↳ non fa modificare il contenuto del segnale ma  
se termina / core dump non fa più eseguire quel contenuto

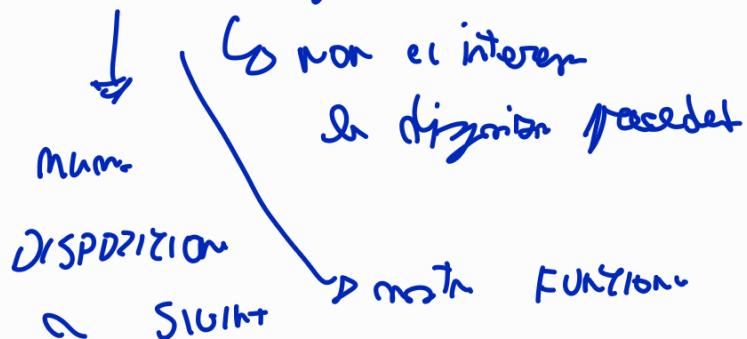
## SIGNAL

(int sig, struct sigaction \*act, struct sigaction \*oldact);

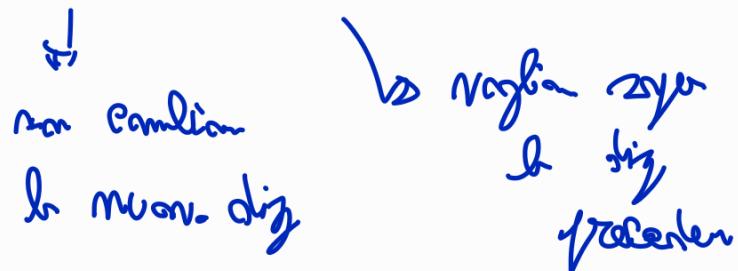


es  
Subtraction (zig, Nuc, older) mit der alten (Nuc)  
  
 (non combined)

## SIGNACTION (SIGINT, SIGTERM, NUL)



## SIGNACTION (SIGINT, NUC, DOLD\_NUC)



## STRUCTURE SIGNACTION

1) VOID (\*SA\_HANDLER)(INT)

↓

punt ≈ Fun

→ SD\_HANDEL definisce il contenuto del Regno  
tramite una funzione utente / SIG-IGN (ignora)

SIG - DEL → default

2) SD\_MASK SIG SET-T

→ initializza i regni alla libreria

3) SD\_FKBS INT

↪ oppure aggiornare

## SIGEMPTYSET

↪ gestisce delle MASK

→ se la SIGEMPTYSET ha la MASK non definita:

SIGCHPYSSET (dalla SD-MASK) → mette regne  
non libere

## IGNORE SIGNAL (int SIG)

↪ prende un  
regno

1 STRUCT SIGNACTION SA;

↳ DEFOLI SI "CHIAMO" LA STRUTTURA  
SIGACTION

2 SA.SA\_HANDLE = SIG\_IGN

↳ si definisce la funzione del comportamento del  
segnale → SIG\_IGN (ignora)

3 SA.SA\_FLAGS = 0 → Non ci sono  
Flags

4 SIG\_BLOCKSET (δ SA.SA\_MASK)  
↳ Città la maschera (0)

5 SIGNACTION (SIG, δ SA, NULL);



SI MODIFICATA IL COMPORTAMENTO

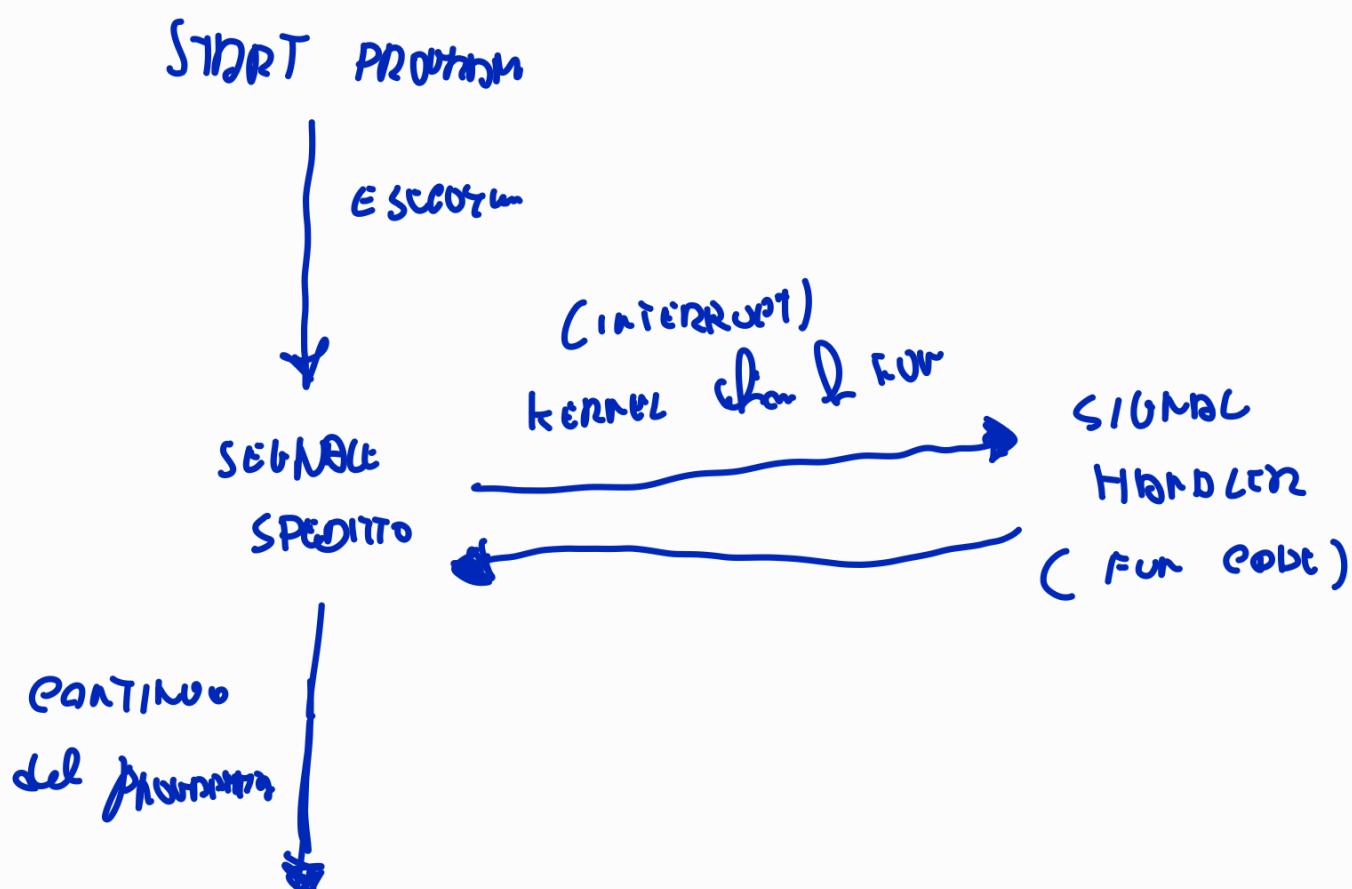
DI SIG SUL FINE LA FUNZ SA



SIG\_BLOCK

# HANDLERS

- funzione definita dall'utente per i segnali
- Quando il segnale viene ricevuto il kernel lo invoca



## ESEMPIO

SIGHANDLER → PRINTF ("OUCH")

1) struttura della struttura SOI (classica)

2)  $S_0 \cdot S_0 - Flags = 0$ ; (nicht flags)

3)  $S_0 \cdot S_b - \text{HANDLER} = S16 \text{ HANDLER}$

4)  $\text{SIGEMPTYSET}(\partial S_0 \cdot S_b \cdot \text{mask})$

↳ set depth mask (non settabel)

VOL DI RITORNO SIGACTION

{  
0 successo  
-1 fallimento

↳ if ( $\text{SIGACTION}(\text{SIGKILL}, \partial S_0, \text{NULL}) == -1$ )  
    ↓  
    exit  
    ↑  
    sigemptyset

C'è errore

→ quando si segnala in esecuzione,  
esso è messo da default in signal mask

↳ fin di esecuzione

PAUSE

→ int pause();

↳ blocca l'esecuzione e chiude fin a quando

- ma non viene restituito il segnale
- ↳ per rendere inattivo un programma finché non viene restituito un segnale
- algoritmico che il segnale è stato maneggiato  
la routine ritorna → se "ERRNO" viene impostata a "EINTR"

## SIGNAL SET

- Oggi: segnale è identificato da un NUMERO INTERO
  - Quando si lavora con tanti segnali è importante
  - per gestirne e manipularne gli INSIEMI DI SEGNALI
- ci sono diverse funzioni di maneggiamento per SIGNAL SET

## → Vediamo solo SIG\_BLOCKSET

### SIG\_BLOCKSET

↳ inizializza a vuoto il SIGNAL MASK

↳ in modo che in genere che nessun segnale venga maneggiato

## SIGNAL MASK

→ un processo ha un signal mask

→ se niente "chiamati" un segnale bloccato, rimane in sospeso fino a quando non viene rimossa dalla mask

## SIGNAL HANDLERS

→ NON CI SONO ESCLUSI DA

→ il main prothibisce a id SIGNAL  
HANDLER

accadrà alle stesse mani che già

## SIGKILLS

→ I segnali NON SI METTONO IN PEND

→ anche se niente generato già sentono

Si ri: contrassegna SOLO UNA VOLTA come PENDING

## ESEMPIO

→ SIGCHLD, generato quando il padre apre il figlio termina

Allo stesso modo si può fare con altri segnali

↳ mettere il handler su SIGCHLD nella sigset

↳ SIGCHLD è BLOCKED

→ mettiamo così che dei figli termini mentre l'handler esegua

↳ solo uno sarà processato

**SOLUZIONE:** SIGCHLD cich e controllerà  
ne un figlio alla volta.

↳ SIGNAL HANDLER qui interrompe il programma  
in qualsiasi momento

↳ handeln di mem sono due flussi di  
esecuzioni simultanee nello stesso processo

**FUNZIONI RIENTRANTI**

→ se può essere eseguita simultaneamente da più threads

**FUNZIONI NON RIENTRANTI**

modificare le variabili globali / statiche

esempio → GLOBAL e FREE → GLOBAL LINKED LIST  
di free memory block

→ se il main chiamava

FREE, & il SIGNAL HANDLER pure

↳ due thread chiamavano

## ASYNC-SIGNAL-SAFE

↳ funzione "sicura" → RIENTRANTE, non è interrotta dai segnali

↓  
ATOMIC

per un programma sicuro:

+1) assicurarsi di chiamare SOLO FUNZIONI SICURE

↳ ASYNC-SIGNAL-SAFE

2) se ha un problema libera i segnali

che chiama funzioni non SICURE

FUNZIONI SICURE → NON USARE GLOBALI DINAMICI

## SIGNAL TRAMPOLINO

Ora si nega il signal handler,

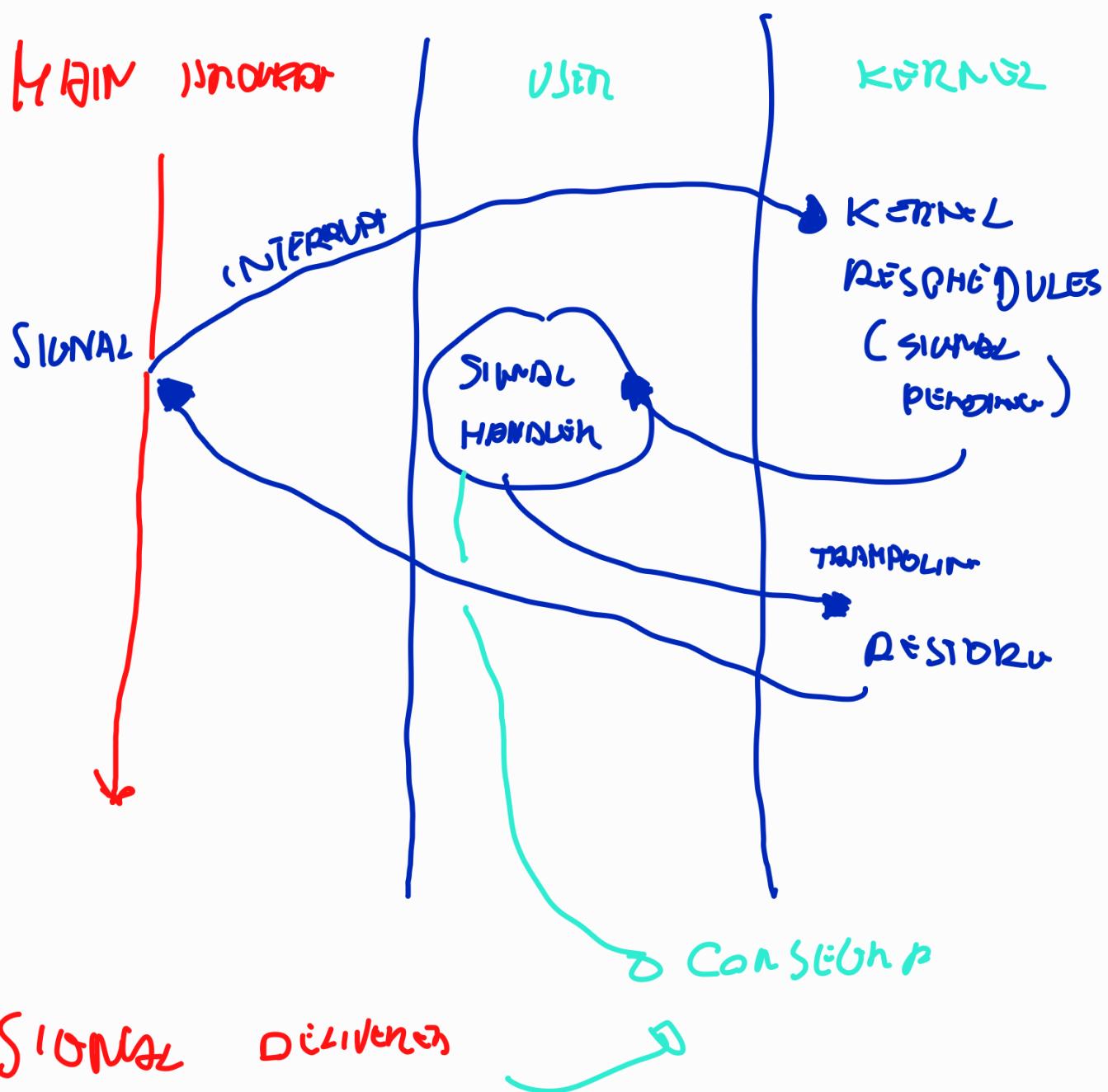
il kernel ricorda alcuni PROCESSI "kernel"

↳ { SIGNAL MASK, SIGNAL STACK  
REGISTER }

→ con miti della funzione SIGNAL HANDLER

↳ SIGNAL IN USER MODE, con faccia  
a modificare il contenuto

↳ SIGNAL TRAP QUOTE



# Comunicazione Asincrona:

Un segnale può essere inviato ad un thread in

quadro temporio durante la sua esecuzione

Esempio: ogni che il kernel gestisce l'interrupt  
e' a sua volta in USER MODE

## EXECUTE SIGNAL HANDLER

### 1) HARDWARE INTERRUPT

↳ schedula fuori dalla CPU

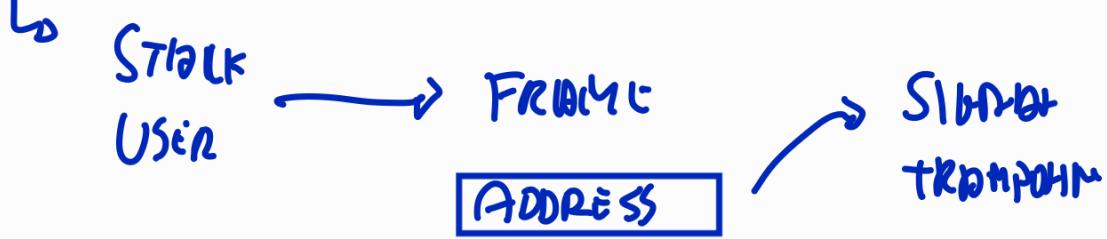
↳ il kernel prende il controllo e ricrea  
i dati del processo

### 2) il kernel sceglie il process che svolgerà il segnale di segnale in PENDING

### 3) fa gestire il segnale, il kernel salva le info del process sulla stack utente

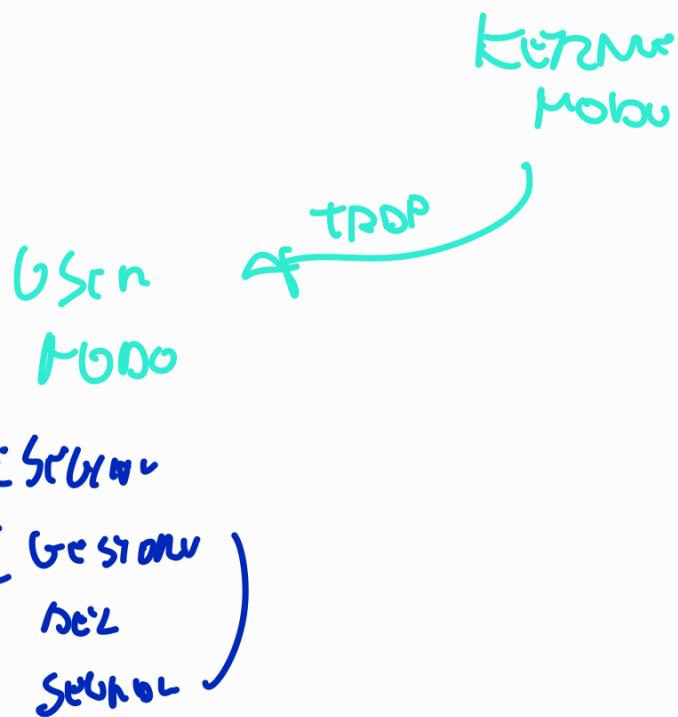
+

PROCESS CONTEXT → CPU REGISTERS, SIGNAL MASK  
ECO



H Quel che giorno tiene un segnale, l'esecuzione  
del process viene interrotta → INTERRUPT

h PREPARE →

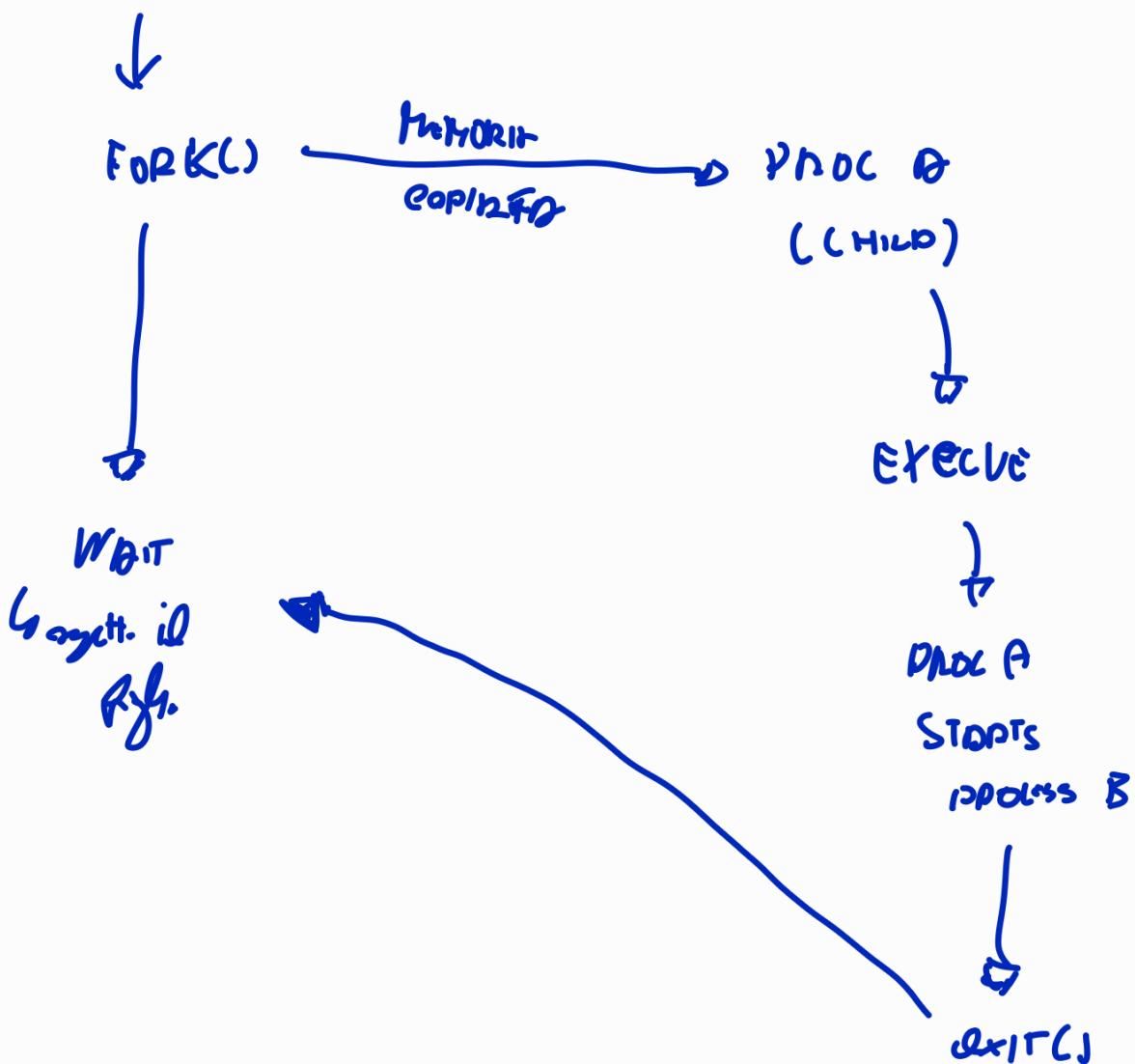


## PROCESS LIFE CYCLE

- SYSCALL CALL
- |   |  |
|---|--|
| { | FORK() → crea un nuovo figlio                                |
|   | EXIT() → termina il process                                  |
|   | WAIT() → aspetta la morte del figlio                         |
|   | EXECL() → lancia UN NUOVO PROGRAMMA<br>nel process esistente |

## OSING SYSTEM CALL

PADRE PROC A



## FORK

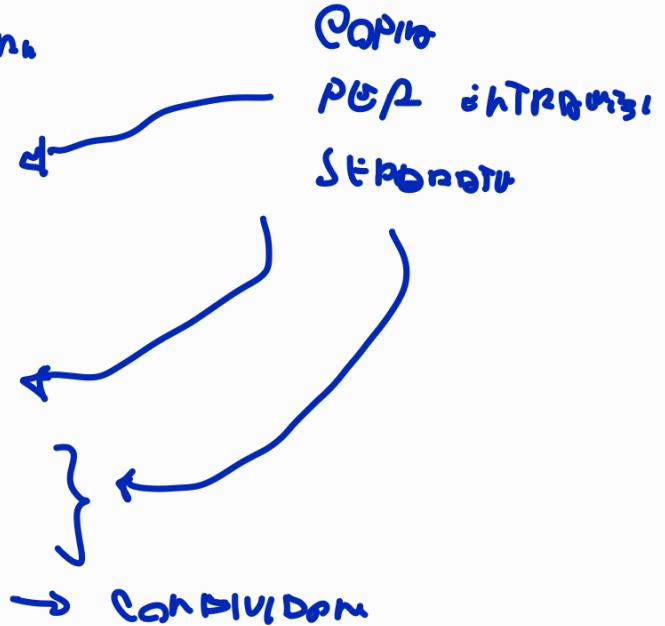
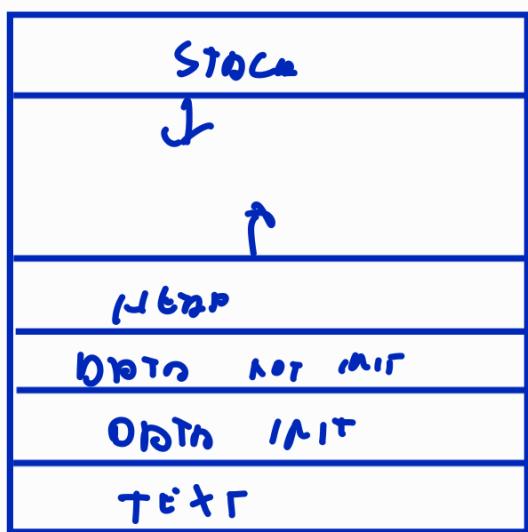
PID = `FORK();`

↳ crea un duplicato quasi esatto del padre

→ Memoria Duplicata → Solo L'è tabella di

POINT DUPPLICATA

## Dai processi nella memoria



Ogni gola è come se fosse INDIPENDENTE

## ESEMPIO 2

→ creare la gestione dei segnali

↳ SIGHANDLER , MANUALLY IMPLEMENTATO SULLO stesso funzione  
SIGALRM - HANDELER

↳ ritorna il pid ricevuto

↳ elaborazione per SIGNAL RM

→ mettere un alarm(z) per generare SIGALRM

→ creare un fork();

if figlio crea la sua classifica CHILD\_SA

↳ vuole info sul SIBOBO

→ vorrei controllare l'hardware del figlio e verificare che  
ci sia qualche problema

→ codice righestrazione stampante

↳ vuole che io scrivendo SIR

Riccardo è visitato OAL

Plinio è FIORE

EXIT → termina il process e avvia il buffer studio

## CHILD process

il process lavora con la WAIT per monitorare  
il contenuto di ciascun dei figli,

il Lavora su diverse informazioni sullo stato dei

Fiori:

- Quale sensore ha registrato o accese il process
- se ha prodotto un core DUMP

## WAIT PID()

→ PRO-T WAITPID (PID\_T PID, int \*WSTATUS, int OPTIONS);

↳ attende che un determinato figlio cambi stato

↳ se non cambia il suo STATO → si blocca

→ continua lo stato → ritorna subito

RITORNO → PID DEL FIGLIO

Se no SUCCESSO

→ re  
eretta

WAITPID(PID, STATUS, OPT)

{  
-1 → qualcun figlio  
>0 → figlio che ha il stat uguale  
==0 → qualcun figlio nella stessa grupp di processi  
<-1 → qualcun figlio nel grupp PID (BBC)

DI DEFAULT option : figli morti

→ 2° param option

ES:

WNOHANG → attende non bloccabile se nessun figlio ha cambiato STATO

**WAIT** ( INT \* status ) ;

→ equivalent = WAITPID (-1, &status, 0) ;

## ORPHAN

→ Un process che non più del padre

↳ esempio: oh INT → PID 7

↳ sarà lui ad eseguire l'arr

ACCEPTED ritorni il PID di INT

## ZOMBIE

→ se un figlio muore prima che il padre faccia

lo mai

↳ il bambino diventa uno zombie

il genitore deve riportarlo vivo o morire

con WAIT

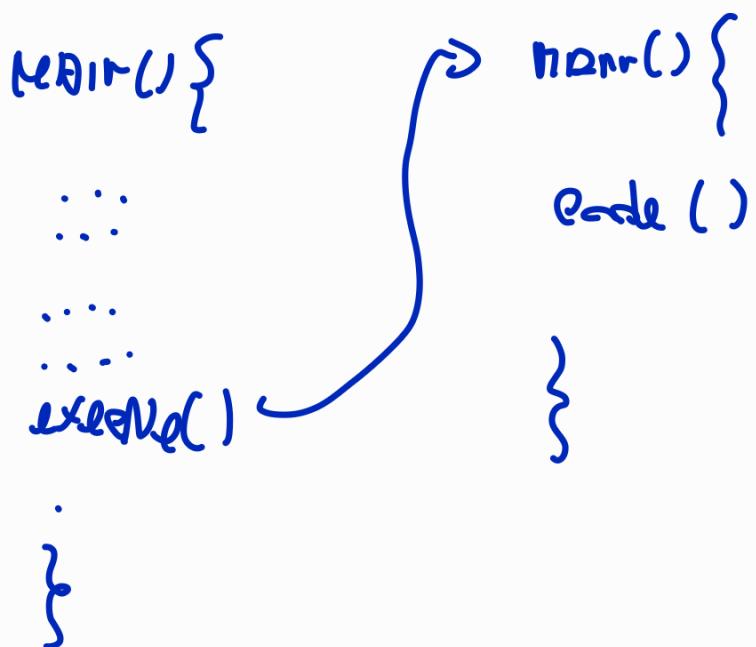
## EXECVE

↳ carica un nuovo programma nella memoria del chiamante

↳ NUOVO → {  
STOCCARE  
OBRAZ

HTTP

→ l'elenco riguarda dall'HTTP



EXECUTE( PATHNAME, ARGV[], ENV[] );

crea il vigen nel percorso assolu./relativ.

ARGV[] → yepif: commenti da linea di comando

ENV[] → NM di ambiente

Una exec che ha successo non ritorna

STACCO

→ **producer** is system call

↳ **producer** has communication TDD technique  
↳ application

→ client of integrate module

→ pull file too

→ request goes to other system calls

**STRUCT** → **STRUCT.COM**

↳ client of return

→  $\mu$  :  $f(x)$

**STRUCT** → **STRUCT.COM**

↳ PID + shared to system

**STRUCT** → **STRUCT.COM** (t)

↳ return code modified

↳ come gray

**Pipe** ⇔ **FIFO**

→ combination of both to green

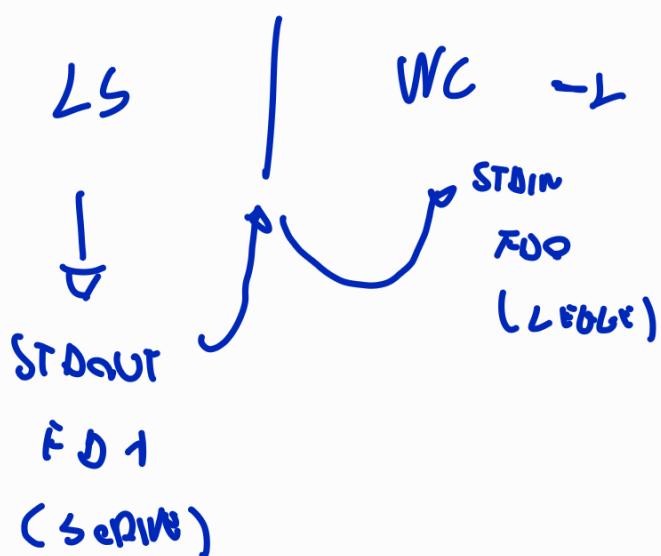
→ queue ; green; children; files etc

Jan niflumt der pipe / fifo, i dati man lett.

Magazin reader

Byte Stream (Data = sequence of bytes)

→ PIPE both → kernel memory

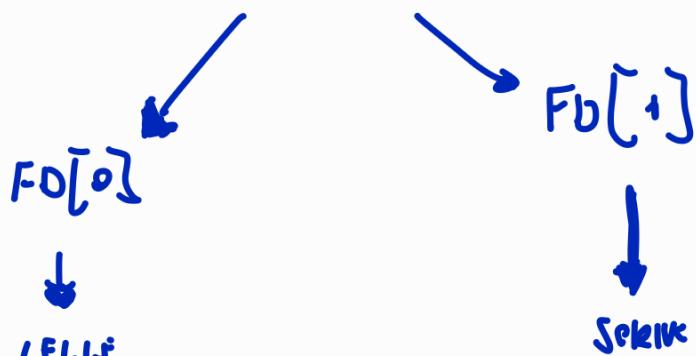


→ Se give leggera libertà di scritt,

sarà VM DIRIGIBILE

→ Hanno un limite

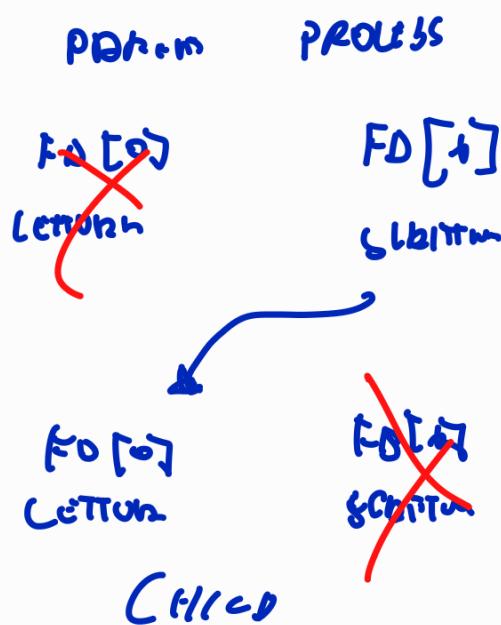
Int PIPE (FILE OUT [→])



- Über
- utilizzate il sistema per  
leggere e scrivere i dati in BYTES
  - Scrivete i BYTES → OPERAZIONE  
DI SCRITTURA → Lettore non è un  
atomo, cioè non è  
possibile accedere  
a più di un carattere  
alla volta.
  - Se non c'è SPazio, nessun BYTE sarà  
scritto
  - connettete la PIPE → fork()

### Übung

trasferire dati su zelle e figli



DUP2 / DUP

↳ Taylor il file descriptor

```
dup2(fd[0], STDIN_FILENO);
```

```
close(fd[0]);
```

## FIFO

↳ come do pipe → non FIFO für um move nel filesystem

in shell → mkfifo -M PERMISSIONS PATHNAME

Int mkfifo(~~const char \*~~ pathname, mode\_t mode)

↳ si ritrovare con UNLINK / REMOVE

```
{ fd = open("ff", O_RDONLY);  
    ↑ lettu  
    ↓  
    fd = open("ff", O_WRONLY);
```

il lancia int della fifo, si trova nel kernel

-> l'idea dell'file system:

FIFO non ha mai un'attesa.

Opponeva però permettere ai processi di creare  
oltre che cancellare.

↪ ciò fa sì che possano partire solo i primi  
le quantità  
rimanenti in rete

## I/O MODULE

RISOL), RISOL) operano su un singolo circuito FO o su  
VOLTE

Si deve così dare modo a possibile I/O  
esistente:

RISOL) può ricevere dati da un dispositivo  
RISOL) che erogherà da parte del dispositivo

↪ se a volto vorremo uscire più di quelli  
di I/O si può fare oppure con  
ogni munito più di