- https://www.cs.unibo.it/~renzo/so/lecture_examples $\boxed{\begin{array}{c}2122\\2223\end{array}}$/
  Catalogo Syscall:
- https://so.v2.cs.unibo.it/wiki/index.php/Il_%27%27catalogo%27%27_delle_System_Call

⇒ Esempi → Il "catalogo" delle systemcall
Esercizi Esperimenti ANNO ↘ Esercizi di "lettura" programmi in C

- https://csunibo.github.io/sistemi-operativi/prove/pratica
- github.com/giulio billi01/University-exercises/

mkdir $\boxed{2021-10-07}$
cd
mv
ls —— -l
-a
-i -- inode
print the inode number of each file.

$ ls -lai
inode drwx.. nomi

ogni volta che creiamo una sottodirectory creiamo un altro riferimento alla cartella genitore (tramite ..)

touch <crea
<cambia data ultima modifica (senza cambiare contenuto)

in UNIX di solito non si mettono suffissi per i file di testo.

contenuto cat > oggi
oggi piove (cntrl+D si scrive ^D)
ln link fisico
ln -s link simbolico

ln vecchio nuovo
ora se facciamo ls -i vediamo che è lo stesso file che ha due nomi.

gedit oggi (Modifico scrivo ancora)
cat oggi → oggi piove ancora
cat oggi new → oggi piove ancora

ln -s vecchio nuovo

rm oggi

echo oggi Freddo > oggi
' (virgolette singole) proteggono tutto
" (virgolette doppie) proteggono quasi tutto
possiamo mettere riferimento a delle variabili con $1
e sostituzione di comando
↑ dollaro

` virgolette rovesciate oppure
$(...)

echo oggi 3 > oggifile
cat oggi file → oggi 3
cat $(cat oggi file) →
→ cat oggi 3 → oggi Freddo

---

noSO.c stdint.h
ha preso le define di:
DDRC PORTC

$\boxed{\text{linguaggio C.pdf}}$ $\boxed{\text{prin C ples}}$

Keywords in C : 32 totali
→ list all keywords in C.

Fibo.c

2021-10-15 laboratorio

esercizi di "lettura"

man 3 fopen

void * "generics"

man qsort quicksort di argomenti generici

Funzione comparazione
int (*compar)(const void*, const void*))

<0 =0 >0
a<b a==b a>b compar(a,b)
ordine crescente ma se invertite
ordine decrescente ... dipende da compar.

#include <ctype.h>
#include <stdint.h>

preprocessore compilatore
↑ gestite da ↑

macro inline Funzione

espanse in ogni punto di chiamata

testo macro "generare al volo nomi di variabili concatenando stringhe..."
non lo potete fare con la inline

inline è una "richiesta gentile"
il compilatore decide se farla inline o no

replicazione sintattica (copia e incolla)
da evitare, meglio funzione

man getopt (1, 3)

dynamic allocation
abbiamo anche noi le nostre bestie nere

man gets
↓ NEVER USE THIS FUNCTION

buffer overflow attack
denile of service attack

molto carina invece la GETLINE
dopo dovete ricordarvi di fare la free

openmem_stream
serve per scrivere un vettore pensato come file

Cirano de Berjerack

man 2 read

man 2 write

man 3 printf

sprintf il risultato lo mette in una stringa

printf = sprintf + write
↓ ↓
Funzione normale elaborazione syscall man 2
comodità ← man 3 Kernel mode
"scritto in corsivo nel esempio ristorante"

mkdir 20211015 cd !$ w1.c

man 2 syscall ←

__NR_write definito in
/usr/include/x86_64-linux-gnu/asm/unistd_64.h

syscalls
- gestione processi
- memoria
- gestione device }
- gestione file } I/O
- gestione filesystem ↔ directory link }
- gestione tempo
- gestione utenti
- gestione rete
- Inter Process Comunication (ipc → due processi che vogliono parlare tra loro)
- debug
- miscellanea
  - nome macchina corrente
  - dominio
  - locale
  - shut down
  - reboot
  - suspend

concetto geniale da inizio tempi
UNIX è separare concettualmente le funzionalità:
- creazione processo fork()
- esecuzione programma exec

fork() crea un processo figlio
a immagine e somiglianza del padre"
è biblico. divisione cellulare

```
getpid()
getppid()
                  ... before ... 451319
if (Fork() )
    printf("%d genitore \n", getpid());
else
    printf("%d figlia \n", getpid());
                                451320
                          ovvero getppid+1
```

wait, waitpid ...
↓
un processo qualsiasi terminato,
ritorna il pid del proc terminato

a queste sono associate macro
- WIFEXITED(wstatus)
- WEXITSTATUS(wstatus)
- WIFSIGNALED(wstatus)
- ...

man 2 wait

mystery.c

cos'è quel wstatus? è lo stato finale del processo terminato.
non è il valore di ritorno ma una maschera di bit che contiene anche il valore di ritorno.

#include <sys/types.h>
#include <sys/wait.h> } da man 2 wait

waitpid( pid, &ws , 0 );
↓ ↓ ↓
pid = Fork() nel if ... sopra int ws "options"

underscore
↓
_exit (42) nel figlio

printf("wait ok %d\n", WEXITSTATUS(ws));

+ complichiamoci la vita

man 7 signal (vuole vedere il numero)

grep -r SIGSEGV /usr/include/

_exit è la syscall
exit "per chiudere correttamente ricordati di fare ..., ... e ... cosa" chiama _exit

Il Catalogo delle SystemCall

errore syscall? RETURNS -1 errno
#include <errno.h> variabile

grep -r ECHILD /usr/include/
esiste strerror(errno) #include <string.h>

ora sappiamo creare processi ma non eseguire programmi

FORK hanno MEMORIA PRIVATA padre e figlio quindi le modifiche fatte a una var globale non vengono viste dall'altro.
copy-on-write
se vogliono strutture dati condivise lo devono chiedere esplicitamente e ... CONCORRENZA ...

## 2021-10-22

differenza processi e thread

processo titolare diritti sulle risorse
può avere
1 - più thread fili esecutivi che condividono la memoria.

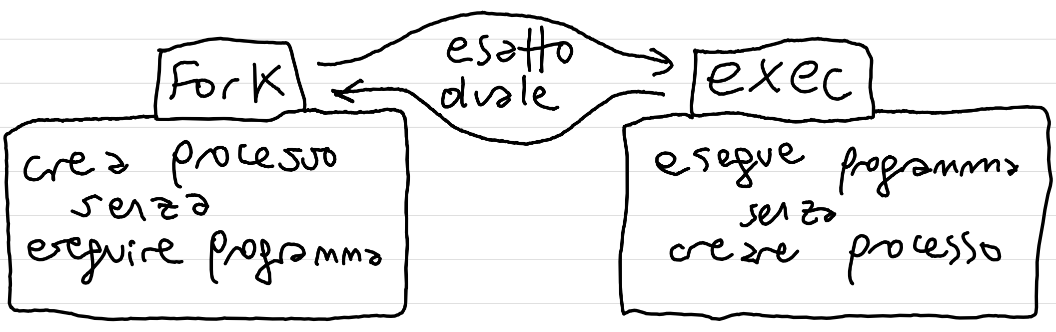
## EXEC

strace

se non lo mettiamo vediamo quello che fa il LINER DINAMICO ← compila con -static
gcc -static -o hello hello.c
strace ./hello
prima di mandare in esecuzione hello.

man 2 exec    man 3 exec

| syscall | suffissi | exec L |
|---|---|---|
| execVE | poi chiamano ← | LP |
| | | LE |
| | | V |
| | | VP |
| | | VPE |



execve (pathname, argv, envp )

pathname: del programma che vogliamo lanciare
argv: vettore di stringhe che vedrà il main del programma lanciato come argv
envp: vettore di stringhe che è l'environment → comando printenv

cosa possiamo lanciare?
eseguibili (esempio $ file hello
ELF ... executable... )
testo con bit eseguibile lo possiamo lanciare con exec
shell di default con elenco comandi.

#! ____
↳ interprete che volete che venga utilizzato per eseguire il nostro file.

testexec.c    testenv.c

man 3 getenv

printenv | grep DISPLAY

xterm →(apre un terminale) export DISPLAY=
e su questo nuovo faccio ① ② → xterm
error can't open display
DISPLAY is not set

man execve    RETURN VALUES, ERRORS

suffissi delle exec →(syscall) execve

Ⓛ long format
argv → numero variabile parametri
execl("..../ls", "ls", "-1", NULL, env);
↳ path file ls : /usr/bin/ls

Ⓥ vettore
char *argv[] = {"ls", "-1", NULL};
execv("..../ls", argv, env);
↳ path file ls : /usr/bin/ls

Ⓟ PATH cercato nelle dir dei comandi nella variabile PATH di environment
il primo parametro diventa "ls" invece di "/usr/bin/ls".

Ⓔ environment

perror

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait>
int main(int argc, char *argv){
        int status;
        printf("%d\n", getpid());
        switch( fork()) {
            case 0: // figlio
                execvp(argv[1], argv+1);
                perror("exec");
                return 0;
            default: // padre
                wait(&status);
                printf("done\n");
                break;
            case -1:
                perror("fork");
                exit(-1);
        }
}
```

## Gestione File

open - close
read write

copia.c
usa system call

fcopia.c
usa libreria c → syscall

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h> // O_... sono definite qui
#define BUFLEN 8192
int main (int argc, char* argv[]){
    char buf[BUFLEN];
    ssize_t n;
    int fin = open(argv[1], O_RDONLY);
    int fout = open(argv[2], O_WRONLY |
            O_CREAT | O_TRUNC, 0666);
    while((n = read(fin, buf, BUFLEN)) > 0) {
            write(fout, buf, n);
    }
    close(fin);
    close(fout);
}
```

**2021-10-29** riassunto:

∀ syscall ha max 6 parametri
passati come 'voci' di memoria
→ puntatori a strutture dati e NON strutture dati.
o interi pK NON entrerebbe in 1 voce di memoria
I parametri devono finire nei registri.
come valore di ritorno:

- quelle che non devono ritornare se non 'è andato a finir bene / 'male' ritornano 0
- quelle che ritornano strutture dati passate per indirizzo
- −1 riservato agli ERRORI
  e in errno (variabile di thread) l'errore
  #include <errno.h> (pseudo var. globale)
  vi contiene il codice dell'errore successo.

2 concetti architettura

syscall UNIX: creare processi (Fork)
ed eseguire programmi (execve)
sono due eventi separati.

creazione di un processo (senza eseguire un programma)
semplicemente avviene una
bi FORK azione (per questo si chiama Fork)
dell'esecuzione e quello che veniva eseguito fino a un certo punto da un solo processo da quel punto in poi viene fatto da DUE processi

NON PROVATE MAI a fare un loop di Fork

```
while (1) { Fork(); } o For(;;) { Fork();}
```

si chiama FORK BOMB!
se è un personal computer ve lo pianta
(non si mette un limite ai processi del singolo utente)
progressione $2^n$ dove n sono i passi.
si riconosce il processo padre dal processo figlio per il valore di ritorno:
padre riceve id del figlio.
il figlio riceve 0.
memoria privata copy-on-write
get pid(), getppid()

Esecuzione di PROGRAMMI (non crea processi)
Se la exec ha successo
cessa di esistere il PROGRAMMA precedente
e nello stesso PROCESSO viene eseguito il PROGRAMMA nuovo lanciato dalla exec.
"Il processo continua la vita ma gli viene cambiata la ricetta".

Ci sono varie librerie man exec (3)
per rendere più facile l'uso dell'unica syscall execve.
+ 1 che vi racconto quando siete più grandi
[registrazioni: 2021-10-29 min 10:42] poiché nuova da pochi anni non è standard POSIX
(si riferisce alla EXECS ?
cerca: "Debian Buster #include<execs.h>"
https://manpages.debian.org/buster/libexecs-dev/system_execs.3.en.html
(scritta dal prof) rd235

```
readline    -lreadline   #include <readline/ readline.h > / history.h >
execs       -lexecs      #include <execs.h>
      char * cmd = readline(NULL);      oppure "%  "
       add_history (cmd);
       system_nosh(cmd);
gcc File.c -lreadline -lexecs -o eseguibile
```

execve (pathname, argv, environment).
pathname: file da eseguire
environment: lista di stringhe "nome = valore" → NULL

```
char * newargv[] = { "ls", "-l", NULL };
```

esempi pathname
- ./hello → newargv[1] è "hello"
- comando? (man 1) → which comando es. $which cat →(output) /usr/bin/cat
- /usr/bin/ls ← which ls /usr/bin/ls

con suffisso -p $PATH → "ls"

Cosa viene ereditato di risorse nel Fork e nel exec? dipende dalla risorsa:

| ereditato? | Fork | exec |
|---|---|---|
| pid (identificativo processo) | ✗ no, cambia | ✓ sì |
| memoria | ✓ apparentemente copiata (copy-on-write) | ✗ no "cambia programma" → ricetta cuoco |

read write open close ...
man FSTAT
lessico per nominare le systemcall
prefisso f prefisso l

man 2 chown, chmod ... e tante altre
senza prefisso: pathname indicato
prefisso F: usano il filedescriptor (aperto)
prefisso l: se stiamo facendo riferimento a un link simbolico la syscall (l-) fa riferimento al link e non al file linkato.
perché? Il link dovrebbe essere trasparente
cosa fa STAT (man 2 Fstat)?
vi da le infor. sul file
the stat structure

```
struct stat {
     ...   ...   ...
     ...   ...   ...
     ...   ...   ...
}
```

Il nome del file NON è un attributo del file (quindi non c'è).
è un attributo della directory che lo contiene.
device, inode (numero file), nlink, uid, gid, size, block
nlink: hard
uid: user; gid: group → numeri interi; es -l
size, block: size cnt
mode: tipo di file e modo di accesso
rwx i permessi sono salvati nel modo.
→ dev
rdev (special file)
numero ← nome
si va in /etc/passwd o /etc/grou

atime accesso · mtime modifica · ctime creazione

teststat.c

```
./teststat /dev/tty
 0 5       character device
./teststat /dev/tty1
 0 0       unknown?
```

← device a cui non abbiamo accesso...
non è la stat che ha fallito ma la open

vi /usr/include/x86_64-linux-gnu/sys/stat.h
/bits/stat.h

| File type | permessi | | | |
|---|---|---|---|---|
| 000 000 | 000 | 000 | 000 | 000 |

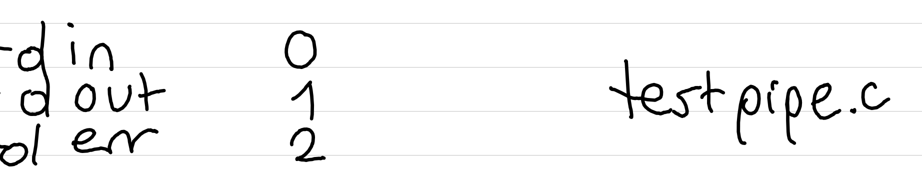
... /bits/stat.h

0 ottale
04 directory
02 ch dev
06 bl. dev
10 reg. file
01 Fifo
12 symb. link
14 socket

---

Inter Process Comunication
ma in realtà crea dei file
quindi oltre la open... parliamo di
PIPE e DUP
man 2 pipe

```
int pipe(int pipeFd[2]);
```



stessa numerazione

stdin 0
stdout 1
stderr 2

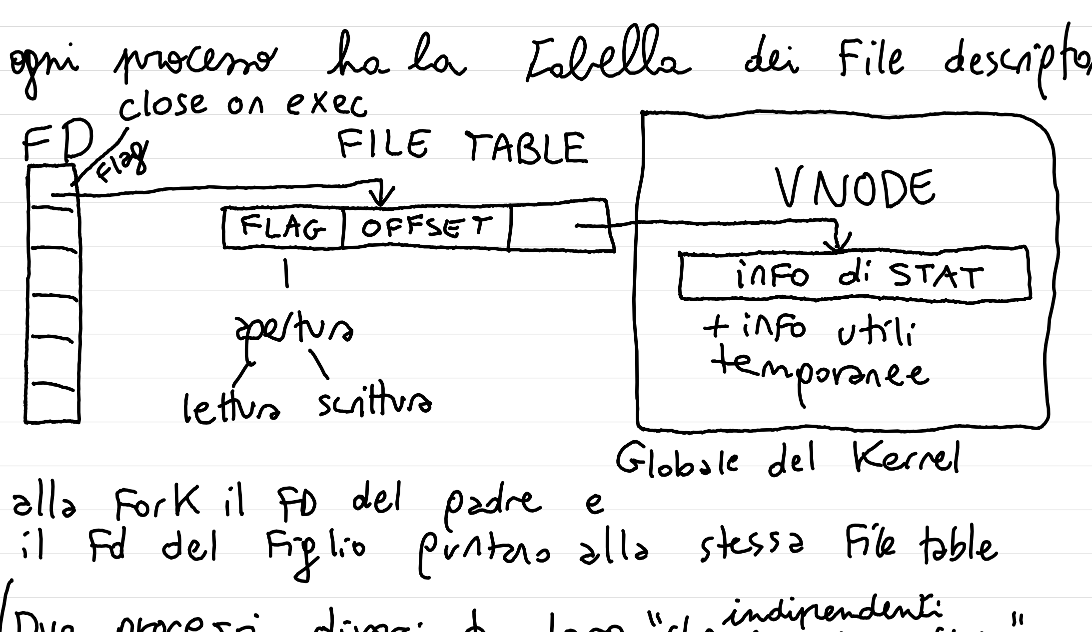
testpipe.c

testredirection.c
DUP dato un file descriptor ne crea un altro che nomina lo stesso file
che se ne fa uno di avere due file descriptor che nominano lo stesso file? Perché talvolta ci sono delle librerie, dei programmi che DANNO SIGNIFICATI PARTICOLARI a determinati file descriptor (esempio STDIN_FILENO = 0
STDOUT_FILENO = 1
STDERR_FILENO = 2)
DUP2
DUP → storica

```
close(STDOUT_FILENO);
DUP(...);
```

questo va bene se non è multithread xk non è atomico.

ogni processo ha la tabella dei File descriptor (FD)



alla Fork il FD del padre e il Fd del figlio puntano alla stessa file table

Due processi diversi tra loro "che indipendenti non si conoscono" hanno ciascuno la propria FILE TABLE
questo perché se il primo legge 100 byte anche il secondo vuole leggere i primi 100 e non i successivi (cambia OFFSET)
→ così non si sovrascrivono tra loro
uno scrive "ciao" l'altro "mare" leggiamo "ciao mare" e non "mare" che ha sovrascritto "ciao"

FLAG IMPORTANTE: close on exec
normalmente ereditati a meno che non abbiamo acceso la close on exec
FCNTL → File CONTROL man 2 Fcntl
oppure argomento della open O_CLOEXEC

textexecpipe.c

readv writev vettore di buffer
consente di avere atomicità
interleaving

pread pwrite offset a partire da qui
LSEEK riposizionarsi all'interno di un file
leggi DESCRIPTION man 2 lseek

IOCTL control device
esempio cambiare volume
cercare web IOCTL_LIST

FNCTL

UMASK

2021-11-05

gestione file system

molto simili ai comandi normalmente usati

so.v2.cs.unibo.it/wiki/index.php/Il_"catalogo"_delle_System_Call

| | |
|---|---|
| xxx | path |
| lxxx | path (link simbolici) |
| Fxxx | File descriptor |
| xxxat | → at calls: |

...at(Fd, path...)

$-D\_\underbrace{\text{GNU\_SOURCE}}_{\text{per O\_PATH}}$

(getdents) di sistema
get directory entries
man 2 getdents64

"These are not the interfaces you are interested in."

man 3 opendir readdir closedir

2021-11-19

Segnali:

signal poll Kill

nocntrlc.c