---------- seti 14/10 ----------

un processo non può interagire direttamente con l'hardware ma per farlo deve usare una system call -> funzione c **wrapper** -> prendono i parametri di una normale funzione in c e poi a seconda della convenzione del sistema operativo preparano i registri ed effettuano la system call

le convenzioni di chiamata per una system call possono essere diverse, tendenzialmente però

- i sistemi assegnano un numero che le identifica, e poi utilizzano dei registri per passare i parametri
- dopo di che si esegue un'istruzione speciale che genera una trap a livello di processore e fa si che l'esecuzione passi a modalità kernel
- il kernel esegue ciò che deve fare
- si ritorna in modalità utente

una system call potrebbe fallire: se hanno successo ritorneranno un valore maggiore o uguale a 0, in caso di insuccesso -1; la ragione del fallimento si ritorna con "errno".
quando si chiama una funzione si deve sempre controllare il valore di ritorno -> per sapere se la funzione è andata a buon fine

una system call è quindi molto più costosa di una normale funzione

perror -> print error

strenerror_r -> traduce da int a string l'errore (decisa da noi in precedenza)

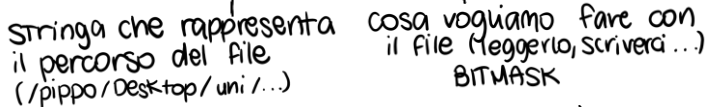
le system call, per portabilità, restituiscono dei tipi che molto spesso sono degli alias a dei tipi già esistenti

tutto l'input/output avviene tramite i **file descriptors** :

- 0 -> STDINT_FILENO -> cin
- 1 -> STDOUT_FILENO -> cout
- 2 -> srderr_FILENO -> cerr

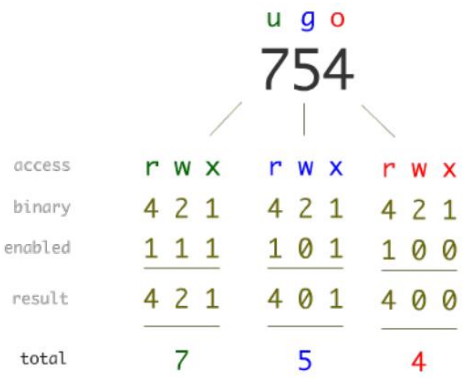
se voglio leggere o scrivere un file. devo per prima cosa aprire il file ->

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



BITMASK -> non è importante il valore completo, ma ogni bit può significare qualcosa
se il secondo parametro (flags) specifica che vuole creare un file nuovo (con il flag create o createfile), bisogna specificare un terzo parametro che rappresenta la BITMASK dei permessi del file

Sotto Unix ogni file ha i permessi relativi a tre categorie di utenti -> il proprietario del file, il gruppo a cui appartiene il file e tutti gli altri utenti del sistema



Per ognuno di questi insiemi di utenti, può specificare 3 bit : r w x -> r: lettura, w: scrittura, x: esecuzione -> spesso codificati in ottale -> 0 ottale, 0x esadecimale

root (amministratore di sistema) può leggere, scrivere ed eseguire (purché ci sia almeno un bit x settato) qualsiasi file -> non si fanno controlli sui permessi

Un problema dei programmi in C e C++ è il memory leak -> valgrind

ctrl-c -> uccide il processo, quindi se si stanno controllando gli errori, non vengono correttamente segnalati

ctrl-z -> sospende un processo, non lo termina

come leggere / scrivere :

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

indirizzo del buffer dove leggere o scrivere

quanti byte andare a leggere o scrivere

Quando non serve più un file si può chiudere usando una `close (int fd);`