---------- seti 28/10 ----------

se un processo termina, ma il padre non lo aspetta -> diventa uno **zombie**

- il sistema non può liberarsi del processo perché il padre potrebbe ancora voler sapere cos'è successo ecc
- ce ne possono essere parecchi, finché il padre non aspetta il figlio o non muore, il sistema non può dimenticarsene
- consumano un po' di risorse del sistema
- quando il processo termina, al padre viene inviato il segnale SIGCHLD, di default ignorato; il padre può aspettarli ciclicamente, oppure non ignorare il SIGCHLD e fare delle wait per i figli persi

eseguire un programma vuol dire eseguire la system call execve -> esistono altre funzioni di libreria che hanno un'interfaccia diversa ma eseguono sempre alla fine execve

```
int execve(const char *pathname, char *const argv[], char *const envp[]);
// libc:
int execl(const char *pathname, const char *arg, ...  /* (char  *) NULL */);
int execlp(const char *file, const char *arg, ...  /* (char  *) NULL */);
int execle(const char *pathname, const char *arg, ...  /*, (char *) NULL,
                                                    char *const envp[] */);
int execv(const char *pathname, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

il primo parametro (pathname) è il percorso del programma che voglio andare ad eseguire; il secondo argomento (argv[ ]) è l'array dei parametri di riga di comando; l'ultimo parametro (envp[ ]) sono le variabili di ambiente

come funziona la redirezione dell'input/output :

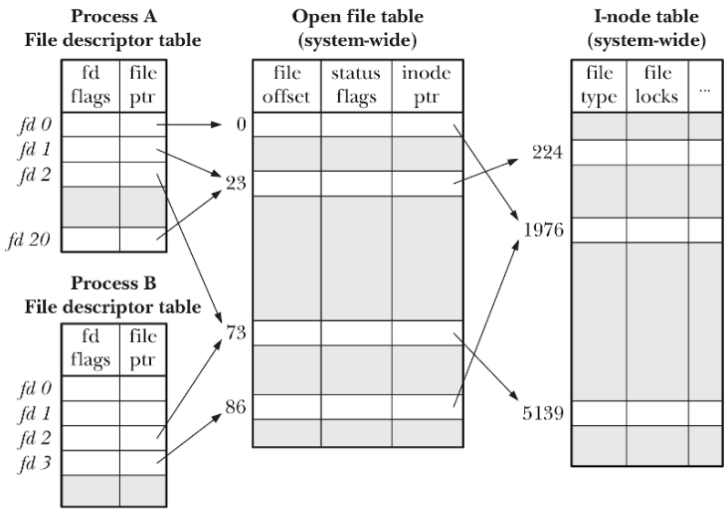

Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

quando apro un file, il kernel restituisce un file descriptor, un intero non negativo che corrisponde al file aperto -> per ogni processo il kernel ha una tabella, la **tabella dei file descriptor** -> ci sono due processi, processo a e processo b, con ognuno la sua tabella
quando passo un file descriptor, la system call utilizza il file descriptor come indice all'interno della tabella del file descriptor di quel processo
per ogni entry nella tabella ci sono due informazioni -> una colonna flag e un puntatore ad un'altra struttura dati che corrisponde ai file aperti nel sistema; questa struttura dati ha a sua volta un offset (che si cambia con lseek), altri flag e dei puntatori che puntano ad un i-node -> una struttura dati che corrisponde a un file all'interno del file system
per ogni file che ho su un file system linux, ci sarà un i-node corrispondente, una struttura dati che contiene vari metadati del file (dimensione, permessi, ecc)

esempio:
supponiamo che il file quiquoqua è l' i-node 224

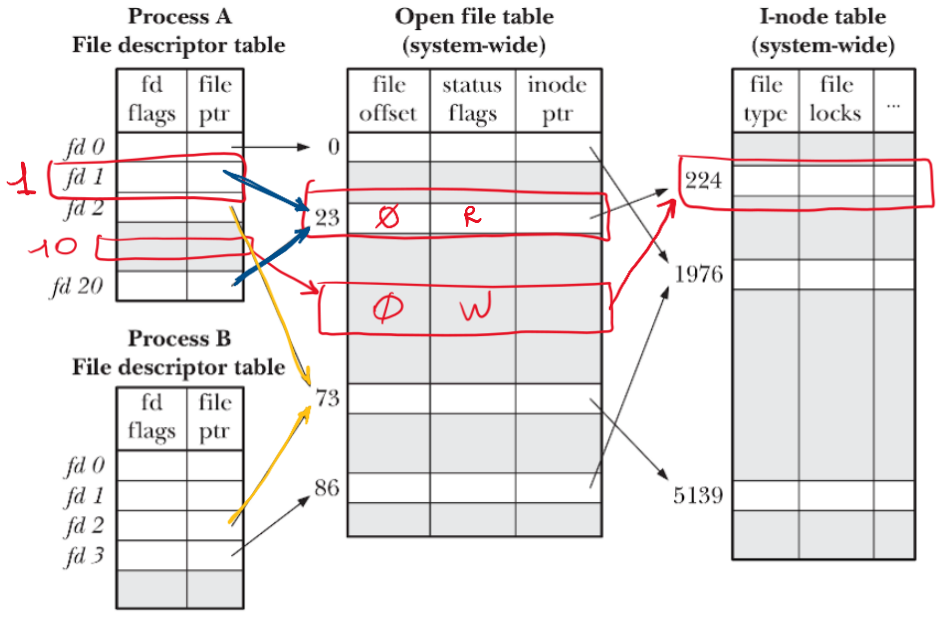

Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

se un processo apre il file che corrisponde a i-node 224, verrà creata una entry nella open file table -> l'offset iniziale è 0, la flag spiega il modo in cui è aperto il file (nell'esempio solo lettura, read) -> viene creata una entry nella tabella del file descriptor -> verrà restituito quel file descriptor al processo; se tutto va a buon fine, la open risponderebbe nell'esempio 1

se un altro processo o anche lo stesso processo fa una open dello stesso file, allora l'i-node sarà uguale, ma non è detto che l'offset e i flag siano uguali -> quindi avremo altre entry in open file table, e corrisponderanno a file descriptor diversi

un'altra possibilità è avere dei file descriptor diversi che fanno riferimento alla stessa entry della open file table (esempio foto fd 1 e fd 20) -> la system call dup crea un altro file descriptor per un file che abbiamo già aperto

un secondo caso è quello dell'esempio delle foto con fd 2 e fd 2 di entrambi i file descriptor -> probabilmente il secondo processo è stato creato tramite fork del primo processo, quindi eseguono le stesse cose, ma dopo la fork ognuno ha continuato diversamente in maniera dipendente

un file descriptor può essere duplicato con **DUP**

- dup restituisce un altro file descriptor che però punta allo stesso file del file descriptor duplicato
- restituisce il file descriptor più piccolo disponibile

quando usiamo la bash (esempio comando ls) l'output del comando finisce sul terminale -> l'output di ls finisce su standard output quindi su fd 1 e fd 1 è collegato al terminale (quindi nell'esempio di prima, il fd originale ha come output il terminale e l'altro il file)