Descrittori “speciali” ed eredità di descrittori

martedì 18 aprile 2023 18:29

Descrittori “speciali” ed eredità di descrittori

Quando facciamo una fork stiamo clonando anche la tabella dei file descriptor, quindi accade che:

Tutti i descrittori vengono ereditati da un processo figlio generato da una `fork()` ⇒ **sharing del file pointer**

Quindi abbiamo una condivisione circa le sessioni già aperte del padre.

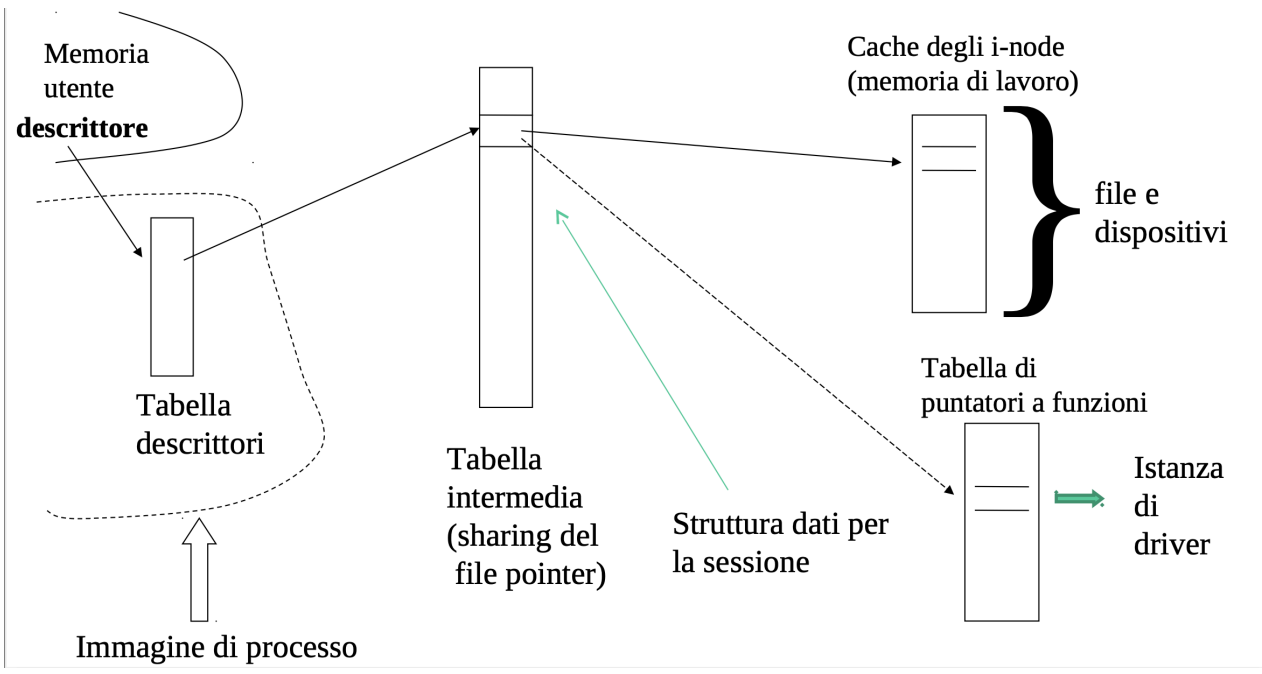
0	standard input	⇒ • associati a specifici oggetti di I/O ed utilizzati da molte funzioni di libreria standard (e.g. `scanf()`/`printf()`)
1	standard output	
2	standard error	• i relativi stream possono essere chiusi

Tutti i descrittori (inclusi 0, 1 e 2) restano validi quando avviene una sostituzione di codice tramite una chiamata execX() eccetto che nel caso in cui si specifichi operatività close-on-exec (tramite la system-call `fcntl()` o il flag `O_CLOEXEC` in apertura del file)

11:44

Il canale 0 crea una sessione per acquisire dati da una tastiera. Esso punta alla tabella intermedia che ci crea una sessione diretta verso la tastiera.
Il canale 0 è a tutti gli effetti un descrittore.
Quindi tutte le applicazioni che richiedono di acquisire dati passano per il canale zero?
Se faccio close(0), scanf non la posso più utilizzare.
Con close posso chiudere i canali e anche i canali standard.

Poi ci possono essere driver diversi che gestiscono gli oggetti I/o (file) e a seconda del driver che gestisce posso avere o non avere il residuo.
Quindi supponiamo di avere due maniglie che creano ad una sessione ad un file 1 e ad un file2, bene: quindi una entry della tabella intermedia ha una gestione di un driver e un'altra entry ha un'altra gestione di un altro driver, quindi su una entry posso avere un problema di residuo o meno.
Poi se ho un'applicazione che crea un figlio, il figlio ereditando la sessione, se il padre era gestito da un certo driver, anche il figlio sarà "adottato" da quei driver.



Il processo child ha la copia di tutte le sessioni raggiungibili che aveva il processo padre.

Scanf è una libreria applicativa che deve consegnare i dati che arrivano da qualche oggetto di I/O. Scanf internamente chiama la sys_call read() e passa un canale, ossia il canale 0, che implica dire che sulla tabella di questo processo attivo, andiamo nella zeresima entry della tabella dei descrittori che va sull'oggetto di I/O, ossia la tastiera.
È fondamentale ricordare che tutti questi descrittori/maniglie sono registrati dentro la tabella dei descrittori (in una ENTRY) e possiamo avere più descrittori che vanno sulla stessa sessione di I/O. Se due descrittori puntano ad una stessa sessione, e uno lo chiudo, la sessione rimane aperta.
Il canale 1 è quello utilizzato da printf(), quando cerchiamo di emettere un messaggio lo scrive nell'output buffer e prima o poi questi messaggi devono essere mostrati su qualche dispositivo di I/O, printf utilizza il canale di I/O pari a 1, e il VFS che vede file, processerà questi dati e li manderà su terminale.

Ma poi noi questi canali li possiamo chiudere?

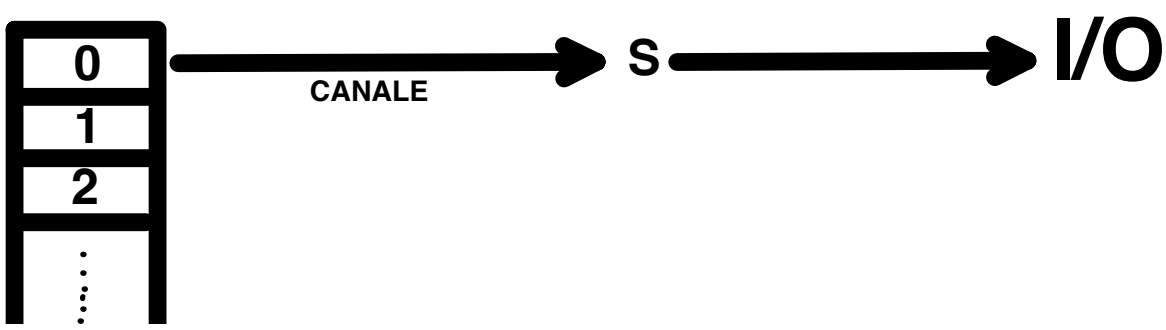


TABELLA DEI DESCRITTORI

Quindi la entry che ci punta sulla sessione che mi porta sul dispositivo di I/O deve in qualche modo essere dismessa, E QUINDI DISMETTERE ANCHE LA SESSIONE, se non ci sono altri canali CHE CI PORTANO verso questa stessa sessione.
La chiusura dello standard input/output si può fare: implica dire ogni volta che andremo ad utilizzare questa funzione di libreria scanf() o printf(), dopo aver chiuso i canali (basta chiamare una close e passare il codice del canale) internamente quzndo queste funzioni cercano di andare sul canale 0/1 per effettuare delle attività (chiamando le read e le write passando opportunamente il canale 0/1) otterremo semplicemente ERRORE.