NAMED PIPE

giovedì 4 maggio 2023 14:39

Named PIPE (FIFO) in Sistemi UNIX

int mkfifo(char *name, int mode)

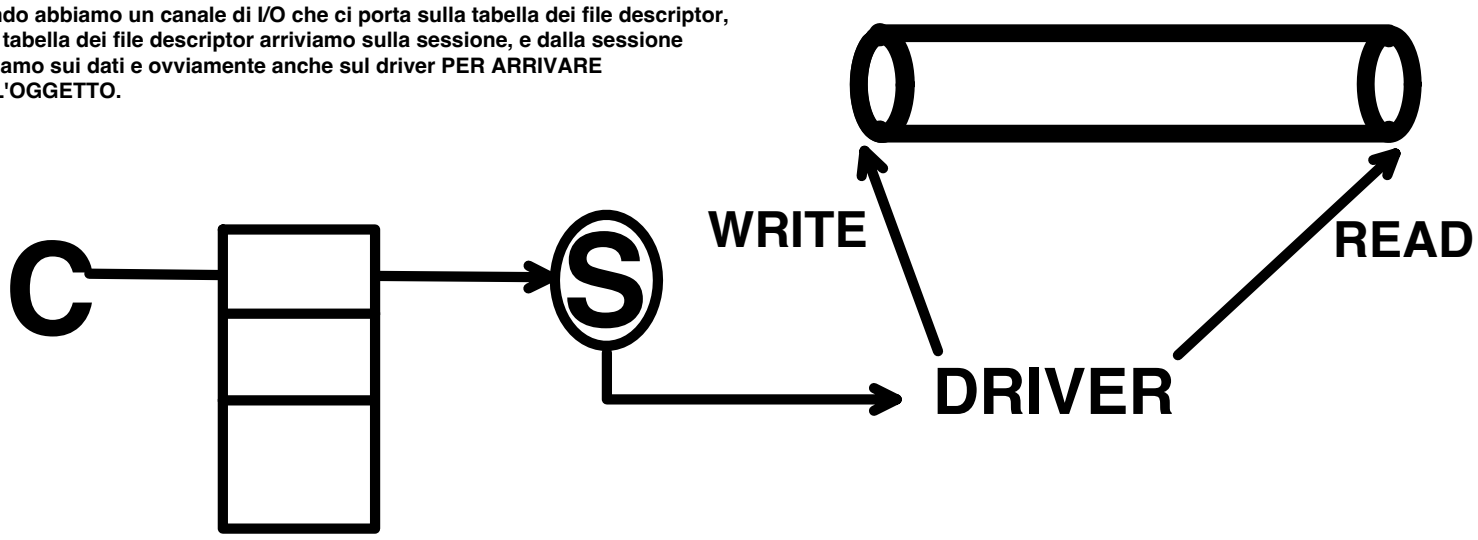
Descrizione invoca la creazione di una FIFO

Argomenti 1) *name: puntatore ad una stringa che identifica il nome della FIFO da creare
2) mode: intero che specifica modalità di creazione e permessi di accesso alla FIFO

Restituzione -1 in caso di fallimento

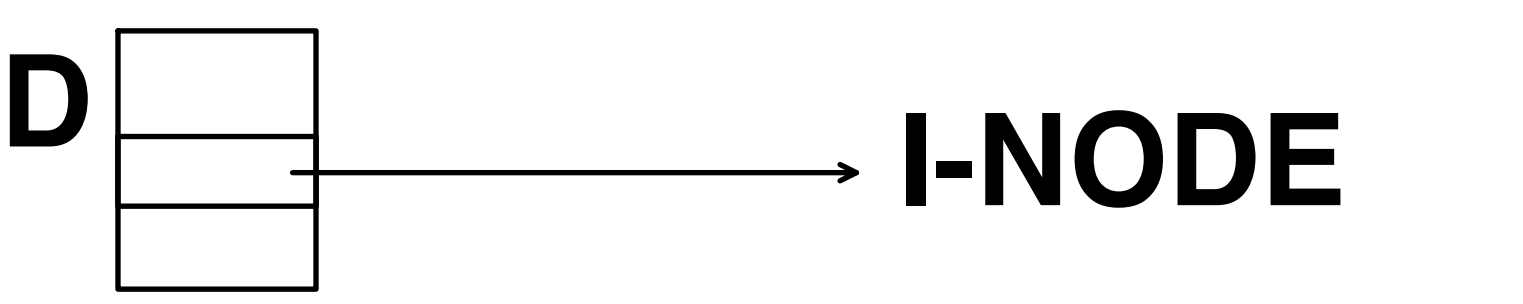
La Named Pipe è un oggetto similare ad una PIPE - stiamo progettando un elemento di comunicazione tra processi/thread, dove noi inseriamo un flusso di byte che può essere estratto dall'estremo di destra. Abbiamo una politica FIFO per quanto riguarda gli inserimenti e le estrazioni di queste informazioni, non ci possiamo riposizionare, se le informazioni le estraiamo queste vengono ad essere cancellate, però questo elemento ha un nome: *se qualcuno crea questo oggetto con quel nome, qualcun altro può aprire questo oggetto e utilizzare quel nome, in particolare per l'operazione di apertura*.
È vero che una certa FIFO NAMED PIPE è tale per cui noi possiamo costruire questo flusso di informazioni con un ingresso da una parte e un'uscita dall'altra, però le reali operazioni che noi facciamo su questa named pipe le facciamo con un unico canale di I/O. Non è un problema se noi abbiamo un solo canale perché nel driver ci sarà una funzione di lettura/scrittura implementata, tale per cui, se su questo canale noi chiamiamo una lettura il driver utilizzerà la funzione interna di lettura che andrà sull'estremo di destra della pipe, mentre la write per inserire i dati andrà sull'altro estremo dell'oggetto.

Quando abbiamo un canale di I/O che ci porta sulla tabella dei file descriptor, dalla tabella dei file descriptor arriviamo sulla sessione, e dalla sessione arriviamo sui dati e ovviamente anche sul driver PER ARRIVARE SULL'OGGETTO.



Le estremità differenti di una FIFO all'interno di un sistema operativo, sono implementate all'interno del driver e quando noi chiamiamo una read e nel driver viene chiamata un'operazione di read, internamente al driver viene riconosciuto un estremo corretto per andare a leggere le informazioni, quindi in particolare l'estremo di questo oggetto di I/O per andare in qualche modo a prelevare queste informazioni da consegnare in lettura.

SULLA FIFO, a differenza delle PIPES, noi lavoriamo utilizzando un solo canale che è il codice che ci viene ritornato dalla system call che noi possiamo chiamare per creare questa FIFO: MKFIFO!
Essa prende in input il nome della FIFO ed eventualmente specifichiamo i parametri per questa creazione o apertura di una FIFO già esistente.
Possiamo specificare i parametri nel caso noi andiamo effettivamente a creare questo oggetto.
Questa named pipe è comunque un oggetto che è presente (se ha un nome) da qualche parte nel VFS che ingloba al suo interno anche il File System, quindi quando noi vogliamo eliminare questo oggetto lo dobbiamo fare in maniera esplicita chiamando la rimozione di un hard-link verso l'oggetto.
Perché in fin dei conti quando creiamo la FIFO e specifichiamo il nome, creiamo una FIFO associata ad un certo hard-link, e questa FIFO è raggiungibile con un certo nome all'interno di una certa directory, ma quando andiamo ad aprire esattamente quell'oggetto la directory ci dirà questo, già lo sappiamo:
La directory dove questa FIFO è presente, quindi il file speciale associato alla directory, avrà una entry al suo interno che ci dice "guarda c'è un oggetto con un certo nome qui, l'I-NODE è QUESTO", e nell'i-node noi sappiamo che possiamo identificare se siamo NON un file ma un oggetto diverso, e quindi una pipe col nome.



Quindi in questo caso tramite le operazioni che vengono effettuate sull'i-node si va a stabilire qual è il driver da utilizzare quando si aprono i canali che ci portano verso questo specifico I-NODE nelle nostre operazioni di I/O.

La rimozione di una FIFO dal file system avviene esattamente come per i file mediate la chiamata di sistema `unlink()`

Operare su una FIFO

- per operare su una FIFO basta aprirla come se fosse un file regolare

Quindi in fin dei conti basta chiamare una open, passiamo quindi attraverso il VFS secondo lo schema che abbiamo mostrato, e quando chiamiamo la open di un oggetto che è all'interno di una certa directory, il file speciale della directory ci identifica l'I-

NODE, e poi in base a quello che c'è scritto all'interno dell'i-node facciamo il set-up corretto della sessione che ci porta verso questo oggetto, e a noi verrà restituito il canale.

- a differenza delle PIPE sulle FIFO si opera quindi con un unico descrittore

- il driver che nel VFS implementa la logica di gestione della FIFO porterà ad operare
  - sull'estremo di "ingresso" se l'operazione invocata è una scrittura
  - su quello di "uscita" se l'operazione invocata è una lettura

Quando noi cerchiamo di effettuare una open su una FIFO è bloccante: se per esempio noi cerchiamo di aprirla in lettura, questa apertura porta il thread che ha chiamato la open in blocco fino a che qualche altro thread non chiama una open in scrittura.

Sui file non è così, perché quando andiamo in lettura su un file andiamo a leggere il suo contenuto e se il file non ha contenuto ce ne accorgeremo leggendo. Su una FIFO, quando cerchiamo di leggere, se non c'è nessuno che vuole cercare di scrivere, noi veniamo bloccati sull'apertura. Quindi non possiamo realmente ricevere un canale di I/O che ci porta a poter lavorare in lettura su quella FIFO fino a che qualcuno non riceverà un canale di I/O per lavorare in scrittura. È evidente che la regola sia questa, di fatto noi abbiamo che le FIFO sono degli oggetti per far parlare processi tra di loro. Ovviamente avviene la stessa cosa se noi apriamo in scrittura e non c'è nessuno che ha un'apertura in lettura, quando chiamiamo l'operazione veniamo ad essere bloccati sull'apertura e quindi il thread viene messo in stato di blocco/wait e non potrà più riprendere il controllo della CPU fino a che eventualmente qualche altro thread (di qualche altro processo o dello stesso processo) non esegua l'operazione duale che ci serva, per avere che una FIFO sia almeno con uno scrittore e un lettore.

- se si vuole inibire questo comportamento è possibile aggiungere il flag `O_NONBLOCK` al valore del parametro mode passato alla system call `open()` su di una FIFO

Questo ci permette di ottenere il codice relativo al canale e poi opereremo in futuro su questo canale a seconda del tipo di operazione che vogliamo effettuare.

- ogni FIFO deve avere sia un lettore che uno scrittore

- se un processo tenta di scrivere su una FIFO che non ha un lettore esso riceve una notifica di errore tramite segnalazione (SIGPIPE) da parte del sistema operativo