Corso «Sistemi Operativi»
AA 2020-2021

Marco Tessarotto

pipes

pipes

- Le «pipes» («tubature») sono il metodo più vecchio di IPC (Inter Process Communication) su Sistema UNIX, apparso nella terza edizione UNIX all'inizio degli anni '70.

- Le pipe forniscono una soluzione al caso d’uso:
- La shell crea due processi (per eseguire programmi diversi), come fare affichè l'output prodotto da un processo divenga l’input per l'altro processo?

Esempio di utilizzo di «pipes» nella shell

- **# conta il numero di files della directory:**
- **ls | wc –l**
- # wc –l : print the newline counts
- Vedere esempio di clone di wc:
- https://replit.com/@MarcoTessarotto/wcclone

- stdout del primo processo diventa estremità di scrittura della pipe
- stdin del secondo processo diventa estremità di lettura della pipe

pipes

- “a pipe is a byte stream” : una pipe è un flusso di byte
- La pipe è un oggetto gestito dal kernel
- La pipe ha una dimensione massima (PIPE_BUF su Linux: 4096 bytes)
- La costante PIPE_BUF è definita in <limits.h>
- La struttura dei dati scritti/letti in/da pipe è a cura dei programmi
- Pipe è una struttura FIFO (first in, first out)
- la pipe è unidirezionale
- lseek non funziona sui descrittori della pipe
- Chiusura dell’estremità di scrittura della pipe* => EOF su estremità di lettura
- *[in caso di dup() dell’estremità di scrittura, tutti i descrittori devono essere chiusi]

fork e pipe

- processo padre: invoca pipe() ed ottiene due file descriptor: estremità di lettura, estremità di scrittura della pipe
- processo padre invoca fork(); il processo figlio riceve un duplicato dei file descriptor aperti del processo padre, tra cui i due della pipe
- facciamo l'ipotesi che processo padre usi la pipe per inviare dati a processo figlio (naturalmente si può fare il incontrario)
- Esempio: https://replit.com/@MarcoTessarotto/sample-pipes

fork e pipe

- processo figlio legge da pipe, processo padre scrive su pipe
- processo figlio deve chiudere estremità di scrittura della pipe
- processo figlio legge da pipe con system call read(), fino a EOF
- read() blocca in caso di pipe vuota e ritorna in casi di dati scritti nella pipe o in caso di EOF
- processo padre scrive nella pipe con write()
- quando l'estremità di scrittura della pipe viene chiusa (anche le copie del file descriptor) allora viene inviato EOF all'estremità di lettura della pipe

Processo che legge da pipe

- chiude file descriptor dell'estremità di scrittura (se no impedisce EOF)
- Legge da estremità di lettura fino a EOF con system call read()
- Se pipe vuota: read() si blocca; read() «tornerà al programma» quando arriveranno nuovi dati scritti sulla pipe dall'altro processo oppure in caso di EOF (cioè quando estremità di scrittura della pipe viene chiusa)
- Ogni lettura legge al max PIPE_BUF bytes alla volta

man 2 pipe

- Se un processo tenta di leggere da una pipe vuota, read() si bloccherà fino a quando i dati non saranno disponibili.
- Se un processo tenta di scrivere su una pipe piena, write() si blocca fino a quando non sono stati letti dati sufficienti dalla pipe per consentire il completamento della scrittura.
- Il canale di comunicazione fornito da una pipe è un flusso di byte: non esiste il concetto di confini del messaggio.

man 2 pipe

- Se tutti i descrittori di file che si riferiscono all'estremità di scrittura di una pipe sono stati chiusi, un tentativo di read (2) dalla pipe vedrà EOF (read restituirà 0).
- Se tutti i descrittori di file che si riferiscono all'estremità di lettura di una pipe sono stati chiusi, allora una write (2) causerà la generazione di un segnale SIGPIPE per il processo chiamante
- Se il processo che chiama write ignora questo segnale, la system call write fallisce con l'errore EPIPE

man 2 pipe

- Un'applicazione che usa le system call pipe e fork dovrebbe usare la system call close per chiudere descrittori di file duplicati non necessari; ciò garantisce che EOF e SIGPIPE / EPIPE vengano consegnati quando appropriato.
- una pipe ha una capacità limitata. Se la pipe è piena, la system call write si bloccherà o fallirà, a seconda che sia impostato il flag O_NONBLOCK (vedi sotto).
- Diverse implementazioni di Unix possono avere limiti diversi per la capacità delle pipe.
- Le applicazioni non dovrebbero fare affidamento su una capacità particolare delle pipe: un'applicazione dovrebbe essere progettata in modo che un processo di lettura consuma i dati non appena sono disponibili, in modo che un processo di scrittura non rimanga bloccato.

esempi

- Pipe per comunicazione dati da processo padre a processo figlio
- https://replit.com/@MarcoTessarotto/sample-pipes

- esempio: pipe per sincronizzare processi figli con processo padre
- https://replit.com/@MarcoTessarotto/pipes-process-synchronization

- esempio con pipe non bloccante:
- https://replit.com/@MarcoTessarotto/pipe-nonblocking