

Esercitazione [9]

Pipe & FIFO

Leonardo Aniello - aniello@dis.uniroma1.it

Daniele Cono D'Elia - delia@dis.uniroma1.it

Federico Lombardi – lombardi@dis.uniroma1.it

Sistemi di Calcolo - Secondo modulo (SC2)

Programmazione dei Sistemi di Calcolo Multi-Nodo

Corso di Laurea in Ingegneria Informatica e Automatica

A.A. 2015-2016

Sommario

- Soluzione esercizio su EchoServer multi-thread
- Obiettivi dell'esercitazione
- Pipe
- Esercizio: Logger
- Named pipe (FIFO)
- Esercizio: EchoProcess su FIFO

Soluzione Esercizio su EchoServer multi-thread

- Esercizio: completare l'EchoServer in modalità multi-thread
- Soluzione
 - La struct `handler_args_t` richiede:
 - Un campo `int` per il descrittore della socket
 - Un campo `struct sockaddr_in*` per i dati di rete del client
 - In `connection_handler()`, va fatta la `free` della struct `handler_args_t`
 - Era stata allocata nel main thread tramite `malloc()`
 - Richiede prima la `free` della struttura dati con le info sul client, allocata anch'essa nel main thread tramite `calloc()`
 - Creazione thread per gestione connessione
 - Assegnare valori all'oggetto di tipo `struct handler_args_t`
 - `pthread_create() + pthread_detach()` con gestione errori
 - "Preparare" la memoria per le info sul client per la connessione da accettare all'iterazione successiva

Obiettivi Esercitazione [8]

- Implementare comunicazione inter-processo tramite pipe
 - Usando pipe semplici tra processi «relazionati»
 - Usando FIFO tra processi non «relazionati»

Overview sulle pipe

- Meccanismo di comunicazione inter-processo
- Canale di comunicazione unidirezionale
- `int pipe(int fd[2])`
 - `fd[0]` descrittore di lettura
 - `fd[1]` descrittore di scrittura
 - ritorna 0 in caso di successo, -1 altrimenti
- Chiamate a `read()` su pipe ritornano 0 quando tutti i descrittori di scrittura sono stati chiusi
- Chiamate a `write()` su pipe causano `SIGPIPE` («broken pipe») quando tutti i descrittori di lettura sono stati chiusi
 - Nota: vale anche per scritture su socket ormai chiuse!

Esercizio: Logger

- Un logger è un componente software che scrive su un file una serie di informazioni sul funzionamento di un'applicazione
 - Usato in genere per il monitoring delle applicazioni server
 - Fornisce dati su possibili problemi occorsi a run-time
- Scenario
 - EchoServer multi-thread
 - Il server istanzia un processo figlio (*Logger*)
 - Il Logger riceve dati via pipe dal padre
 - Li scrive sul *log file* (aperto in append mode)
 - Il padre reindirizza il proprio canale `stderr` su questa pipe
 - Risultato: ogni messaggio scritto dal padre su `stderr` viene «trasferito» via pipe al Logger e quindi scritto sul log file

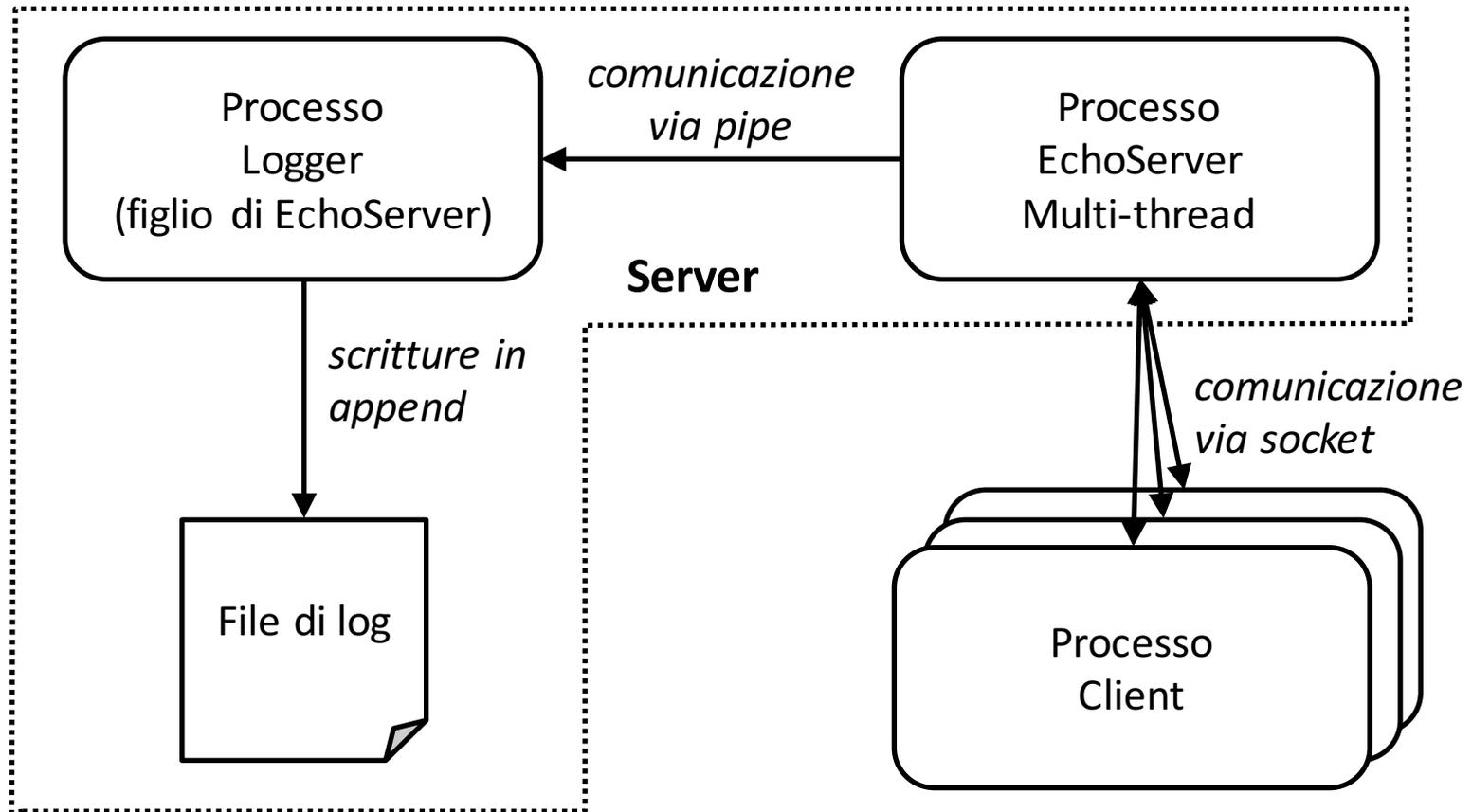
Esercizio: Logger

Come reindirizzare un canale?

- Funzione **dup**: `int dup(int fd)`
 - Effettua una copia del descrittore `fd` usando il primo descrittore inutilizzato (quello con valore *minimo* in tabella!)
 - Ritorna il nuovo descrittore in caso di successo, `-1` altrimenti
- Funzione **dup2**: `int dup2(int oldfd, int newfd)`
 - Come `dup()` ma, invece di usare il descrittore inutilizzato avente valore minimo, usa `newfd`
 - Se `newfd` esiste ed è aperto, viene prima chiuso
- Un canale `fd1` è reindirizzato su `fd2` invocando `dup2(fd2, fd1)`
 - `fd1` diventa un alias di `fd2`
 - Read/write sul «vecchio» descriptor `fd1` sono reindirizzate su `fd2`

Esercizio: Logger

Processi in gioco



Esercizio: completare il codice lato server (EchoServer e Logger)

Overview sulle named pipe (FIFO)

- Simili alle pipe, consentono tuttavia comunicazione tra processi non «relazionati» (nessun legame padre-figlio via fork)
- Una FIFO è un **file speciale** per comunicazione unidirezionale
- **Creazione:** `int mkfifo(const char *path, mode_t mode)`
 - `path`: nome della FIFO (non in uso da altri file)
 - `mode`: permessi da associare alla FIFO (es. `0666`)
 - Ritorna `0` in caso di successo, `-1` altrimenti
- **Apertura:** `int open(const char *path, int oflag)`
 - Nome FIFO e modalità di apertura (`O_RDONLY`, `O_WRONLY`, etc)
 - Ritorna il descrittore della FIFO, `-1` altrimenti
- **Chiusura:** `int close(int fd)`
- **Rimozione:** `int unlink(const char *path)`

Esercizio: EchoProcess su FIFO

- Il server prepara (crea) due FIFO
 - `fifo_echo` per inviare messaggi al client
 - `fifo_client` per ricevere messaggi dal client
- La comunicazione client-server avviene tramite queste due FIFO
- Esercizio: completare codici di client (`client.c`) e server (`echo.c`)

