Sistemi Operativi e Reti di Calcolatori (SOReCa)

Corso di Laurea in *Ingegneria Informatica e Automatica (BIAR)*

Terzo Anno | Primo Semestre

A.A. 2025/2026

Esercitazione [06] Pipe FIFO Socket

Riccardo Lazzeretti lazzeretti@diag.uniroma1.it
Paolo Ottolino paolo.ottolino@uniroma1.it
Matteo Cornacchia cornacchia@diag.uniroma1.it

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA UNIVERSITÀ DI ROMA

Sommario

- Soluzioni precedente esercitazione
- Pipe and FIFO:
  - Esercizio: Produttore/consumatore
- Input/output su socket
  - Esercizio: invio e ricezione dati

Obiettivi

- Imparare ad effettuare comunicazioni tramite architettura client-server su protocollo TCP
- Descrittore di Socket
- Invio/ricezioni messaggi su Socket

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

FIFO: Produttore-Consumatore

- Lab06, es. 1

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Produttore/Consumatore

Lab06 es1: Producer-Consumer con FIFO

- L'applicazione è sviluppata in due moduli separati.
- Si tiene conto della configurazione con `NUM_CONSUMERS` consumatori e `NUM_PRODUCERS` produttori
- Non serve realmente un buffer, ma si usa una FIFO

- Completare il codice dell'applicazione produttore/consumatore
- Sorgenti
  - `makefile`
  - `producer.c`
  - `consumer.c`
- Suggerimento: seguire i blocchi di commenti inseriti nel codice
- Suggerimento: il producer sarà il primo a operare e il consumer l'ultimo
  - Quindi il producer crea la FIFO e il consumer la distrugge
  - La FIFO ha una grande dimensione e si satura difficilemente
    - Ridimensionare la fifo
      - #define _GNU_SOURCE //da mettere a inizio del file producer.c, prima delle include
      - fcntl(fifo, F_SETPIPE_SZ, 10*sizeof(int)); //posizionare subito dopo la open
    - Rallentare il consumer con delle sleep()
- Test:
  - Lanciate prima `producer` (crea semafori e memoria condivisa) e poi `consumer`

Socket

- Invio e Ricezione messaggi

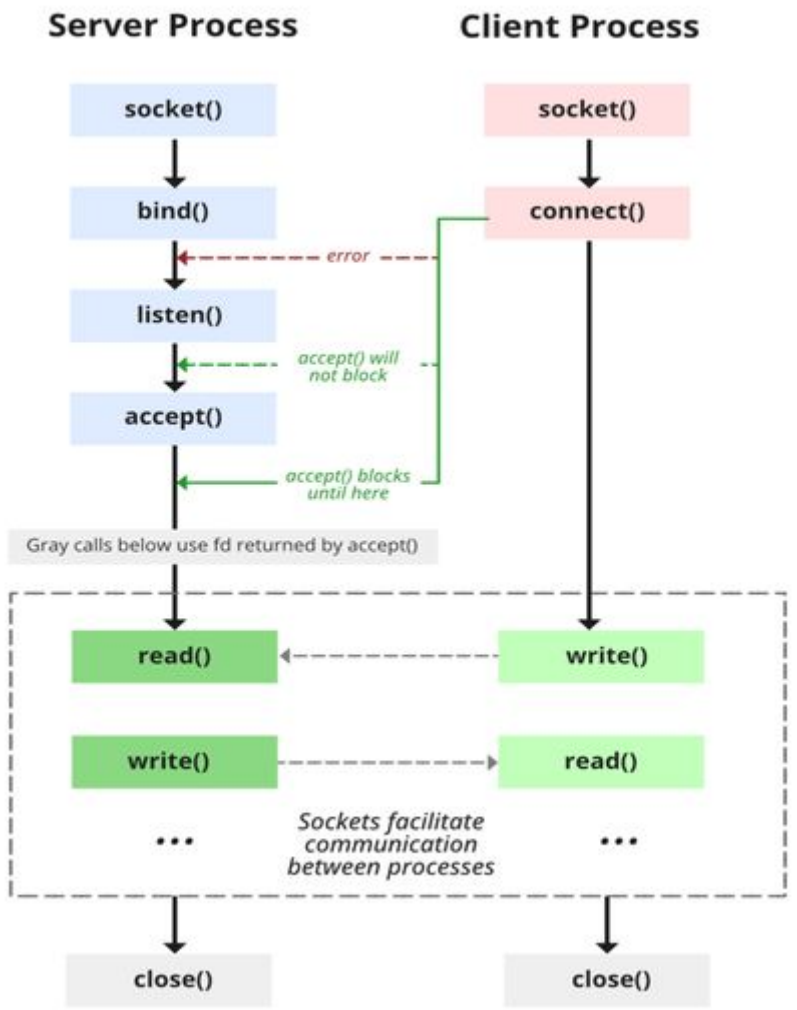
Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Invio e Ricezione messaggi su Socket

Overview

- Scenario: architettura client-server su protocollo TCP
  - Il server è in ascolto su una certa porta nota
  - Il client effettua una connessione verso il server su quella porta
- Una volta aperta una connessione TCP tra due processi, ogni processo può accedervi tramite un descrittore
- L'invio e la ricezione di messaggi tramite socket vengono gestiti in maniera analoga a `write()` e `read()` su file
  - È necessario disporre di un descrittore della socket
  - Lettura e scrittura avvengono a blocchi
  - Ci sono alcune differenze….



Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Invio messaggi su Socket

`send()`

- La funzione `send()` è definita in `sys/socket.h`

```
ssize_t send(int fd, const void *buf, size_t n, int flags);
```

  - `fd`: descrittore della socket
  - `buf`: puntatore al buffer contenente il messaggio da inviare
  - `n`: numero massimo di byte da scrivere
  - `flags`: fissato a `0`, rende la `send()` equivalente alla `write()`

  Ritorna il numero di byte realmente scritti, o `-1` in caso di errore

  - Default: semantica **bloccante**
    - Se buffer di invio nel kernel non contiene spazio sufficiente per il messaggio da inviare, rimane bloccata in attesa…

Ricezione messaggi su Socket

`recv()`

- La funzione `recv()` è definita in `sys/socket.h`

```
ssize_t recv(int fd, void *buf, size_t n, int flags);
```

  - `fd`: descrittore della socket
  - `buf`: puntatore al buffer dove scrivere il messaggio ricevuto
  - `n`: numero massimo di byte da leggere
  - `flags`: se fissato a `0`, la `recv()` è equivalente alla `read()`

  Ritorna il numero di byte realmente letti, o `-1` in caso di errore

  - Ritorna `0` in caso di connessione chiusa
  - Default: semantica **bloccante**
    - Se l'altro endpoint non invia nulla, rimane bloccata in attesa
    - Trasferisce i dati disponibili fino a quel momento nel buffer del kernel, entro il limite di `n` bytes, piuttosto che restare in attesa di ricevere l'intera quantità specificata…

Letture e Scritture su Socket

Valori di ritorno ed interrupt

- L'analisi e gestione dei valori di ritorno per letture e scritture su socket è più complessa rispetto a quanto visto per i file
- La `send()` è analoga alla `write()`: un segnale può potenzialmente causare un invio parziale di dati, o interrompere la chiamata prima che il primo byte venga trasmesso (settando `errno` ad `EINTR`)
- Per la `recv()`, oltre agli stessi effetti derivanti dalla ricezione di segnali visti per `send()`, si pone il problema che al momento della chiamata possono essere disponibili meno dati di quelli attesi!
  - Come distinguere questo caso da quello dell'interruzione dovuta alla ricezione di un segnale?
  - Come fare quando la dimensione dei dati da ricevere non è nota a priori?
  - In questa esercitazione ci limiteremo a gestire soltanto il caso in cui le chiamate vengono interrotte prima che un byte sia stato letto o scritto

Invio e Ricezione messaggi su Socket

Lab06 es2: TimeServer

- Scenario
  - Due processi: un client ed un server
  - Il server è in ascolto in attesa di connessioni TCP
  - Il client si connette al server ed invia un comando
    - Messaggio «TIME»
  - Se il server riceve il messaggio atteso, la risposta conterrà ora e data correnti, altrimenti manderà un messaggio di errore
- Sorgenti: `server.c` e `client.c`
- Esercizio
  - Completare le parti mancanti, relative all'invio/ricezione di messaggi via socket
  - Per l'esecuzione, è necessario lanciare client e server su terminali diversi

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Autovalutazione

- https://forms.gle/txi5dPWeoV7jLLS27

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma