

SIMULAZIONE SERVER DISCORD

Obiettivo:

Simulare il Funzionamento di una Chat Tipo Discord in Locale

L'obiettivo è simulare il funzionamento di una piattaforma di comunicazione simile a Discord, ma in un ambiente locale.

Requisiti Principali:

- **Ogni utente è un processo:** Ogni utente è trattato come un processo indipendente.
- **Indirizzamento dei messaggi:**
 - I messaggi possono essere inviati a un **canale pubblico**.
 - I messaggi possono anche essere inviati a un **utente specifico** in modalità privata.
- **Scrittura e ricezione simultanee:** Gli utenti possono scrivere e ricevere messaggi contemporaneamente.
- **Separazione dei canali:** I canali sono distinti; un utente su un canale non riceve messaggi da altri canali.
- **Comunicazione tramite pipe:** Utilizzo di pipe named o unnamed per la comunicazione tra processi.
- **Scritto in POSIX C:** Implementazione del sistema in C conforme a POSIX.

Idea:

2 programmi principali:

- **SERVER:**
 - Legge i messaggi degli utenti
 - Inoltra i messaggi a tutti (escluso il mittente)
- **UTENTE:**
 - Prende i messaggi in input da tastiera
 - Scrive i messaggi sul server

Tabella dei Componenti



Componente	Descrizione
 Server	Riceve messaggi da tutti, li smista secondo le regole.
 Utente	Processo che può mandare e ricevere messaggi.
 Canali	Ogni canale è gestito dal server. Utenti connessi allo stesso canale si "vedono".
 Pipe	Meccanismo di comunicazione tra i processi. Ogni utente comunica con il server tramite pipe.

Tabella delle Semplificazioni

Aspetto	Scelta semplificativa
Numero di canali	Inizialmente solo 1 canale, generale.
Cambio canale	Chiudendo e riaprendo il processo
Comunicazione	Solo tramite named pipe (mkfifo())
Registrazione utente	L'utente si registra al server passando il proprio nome all'avvio.
Messaggi diretti	Solo messaggi pubblici nel canale.
Numero utenti	Massimo 3 utenti
Server	Unico processo centrale sempre attivo, no interfaccia grafica.
Numero Processi	Ogni utente ha 2 processi figli: uno per scrivere, uno per ricevere.

Comandi usati

- **mkfifo()** → pipe
- **open(), read(), write()** → I/O
- **fork()** → lettura e scrittura simultanea
- **sprintf()** → nome utente

Compilazione

Per compilare i file sorgente C, segui questi passaggi:

1. **Aprire il Terminale:**

Accedi alla cartella contenente i file .c che desideri compilare ed eseguire i seguenti comandi:

```
-----
| gcc server.c -o server |
| gcc utente.c -o utente |
-----
```

Risultato:

Dopo aver eseguito questi comandi, verranno generati i due eseguibili: **server** e **utente**.

2. **Creare** manualmente le **pipe nominate** (FIFO) per ogni utente.

Ad esempio, per 3 utenti chiamati mario, luigi e peach, eseguire:

```
-----
| mkfifo mario_in mario_out |
| mkfifo luigi_in luigi_out  |
| mkfifo peach_in peach_out |
-----
```

Risultato:

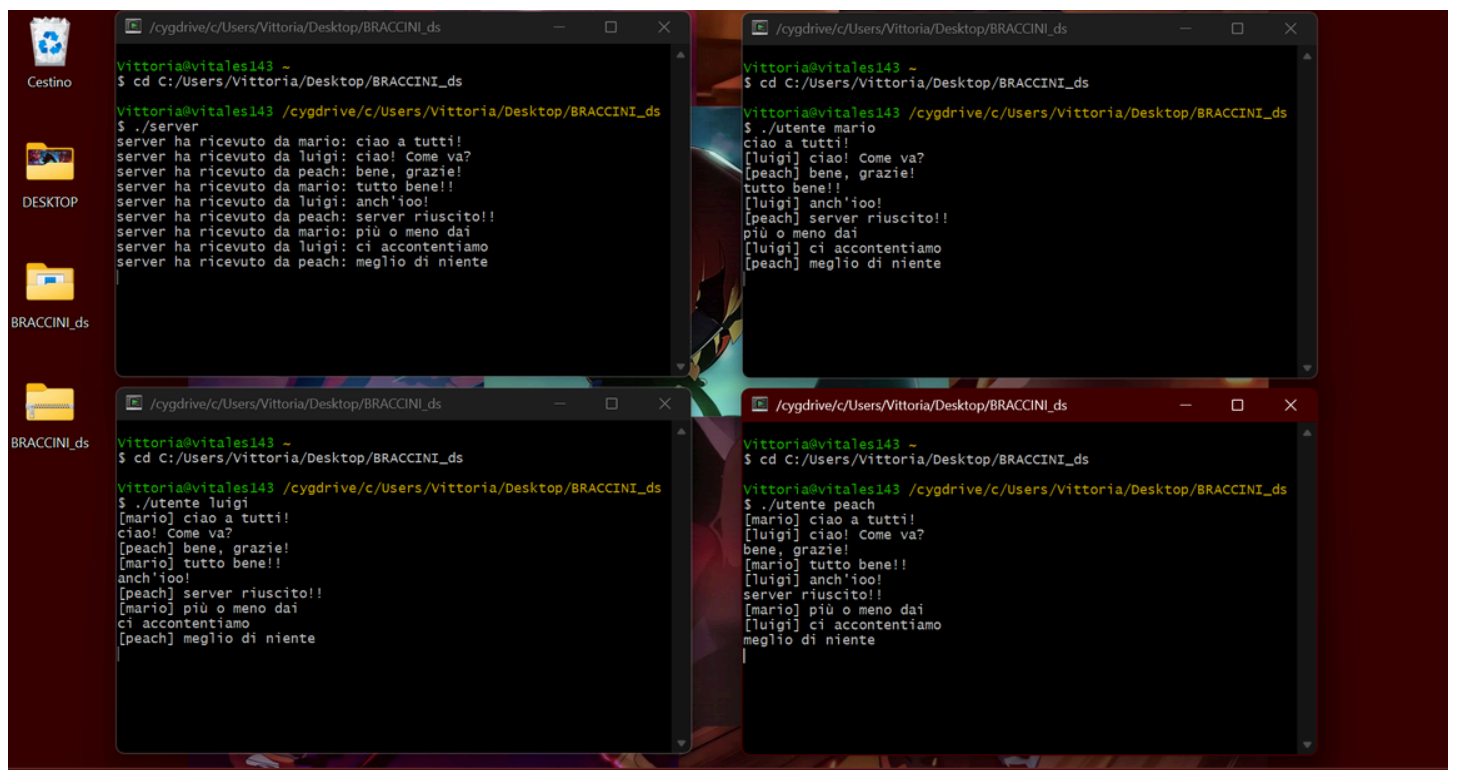
Dopo aver eseguito questi comandi, verranno generate **6 pipe**

3. **Avviare** il server:

```
-----
| ./server |
-----
```

4. **Avviare** tutti gli utenti:

```
-----
| ./utente mario |
| ./utente luigi  |
| ./utente peach  |
-----
```



OSSERVAZIONI:

È normale che l'invio dei messaggi degli utenti possa essere "scombinato": succede perché il **server legge da ciascuna pipe in sequenza**, ma solo **una pipe alla volta** per ciclo, e solo **se c'è qualcosa da leggere**. Per **evitare** questa cosa, si consiglia di far scrivere agli utenti **un messaggio alla volta, rispettando l'ordine**: se sono stati **avviati in sequenza** prima Mario, poi Luigi e poi Peach, il primo messaggio lo scriverà Mario, poi Luigi e poi Peach.