Discord in Locale: Documentazione del Progetto

1. Analisi del Problema

Il progetto richiede la realizzazione di un sistema di chat Discord che funzioni in locale utilizzando meccanismi di IPC di POSIX C.

Requisiti Principali:

- Simulare server Discord in ambiente locale
- Permettere agli utenti (processi) di scrivere messaggi su canali
- Supportare messaggi indirizzati a canali o utenti specifici
- Permettere la ricezione di messaggi mentre se ne scrive uno
- Gestire più canali con isolamento dei messaggi tra canali diversi

2. Progettazione e Descrizione dell'Architettura

2.1 Architettura Generale

L'architettura si basa su un modello client-server:

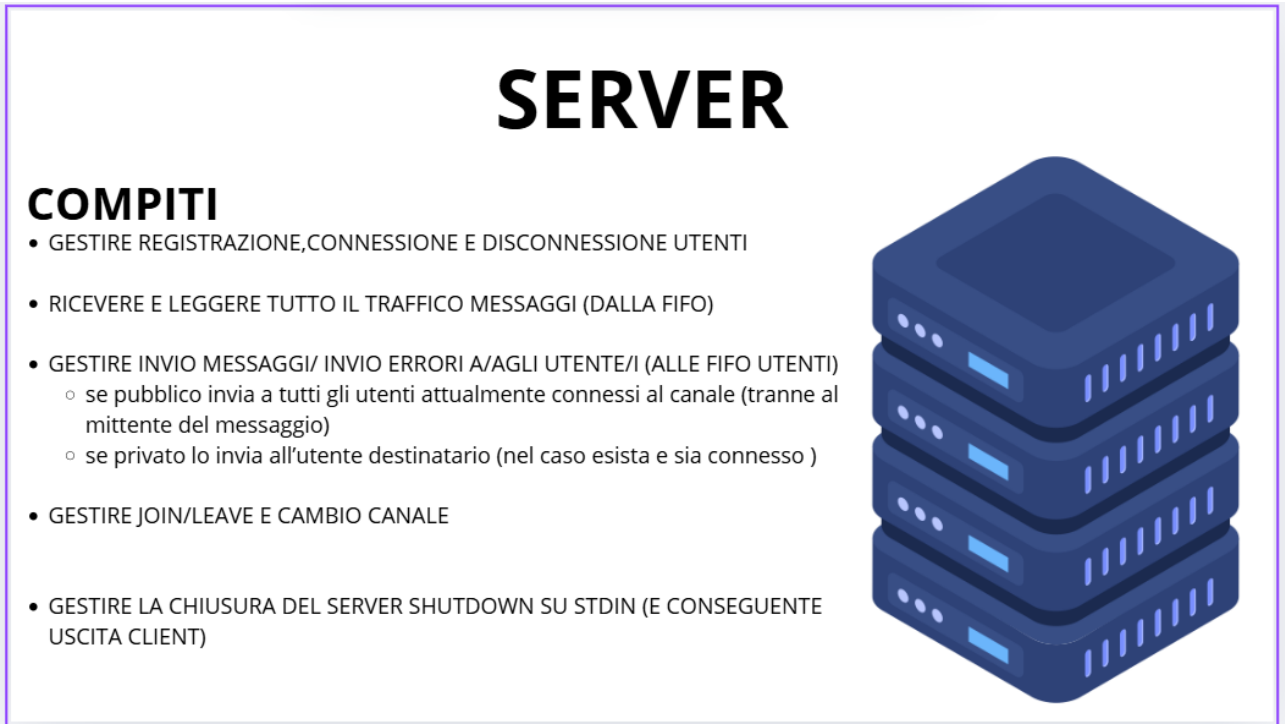
- Il server gestisce la comunicazione tra client e mantiene lo stato del sistema
- I client si connettono al server e inviano/ricevono messaggi

2.2 Componenti Principali

Server: uno solo (gestito su un terminale separato)

- **Handler Child**: Processo figlio che gestisce i messaggi in arrivo e li smista in base al tipo di messaggio

- **Shutdown Reader (stdin reader)** : Processo figlio che monitora i comandi di spegnimento
- **Processo Padre**: Coordina i figli e gestisce l'avvio/spegnimento (comunicando lo shutdown all'handler child)



Client: uno per ogni utente (terminale)

- **Reader Child**: Legge messaggi dalla propria FIFO e li visualizza (comunica anche shutdown al writer)
- **Writer Child**: Legge input utente e invia messaggi al server (legge anche la unnamed pipe per ricevere avvenuta connessione e shutdown)
- **Processo Padre**: Coordina i processi figli

CLIENT

COMPITI

- gli utenti creano la propria fifo privata dove leggere
- gli utenti si registrano/connettono segnalandolo al server (anche canale a cui appartengono)
- una volta registrati possono parallelamente:
  - leggere/stampare i messaggi in arrivo (tranne quelli che invia il mittente)
  - scrivere i messaggi privati o quelli pubblici del canale
- gli utenti possono disconnettersi (o cambiare canale) (volutamente o per shutdown server)

4

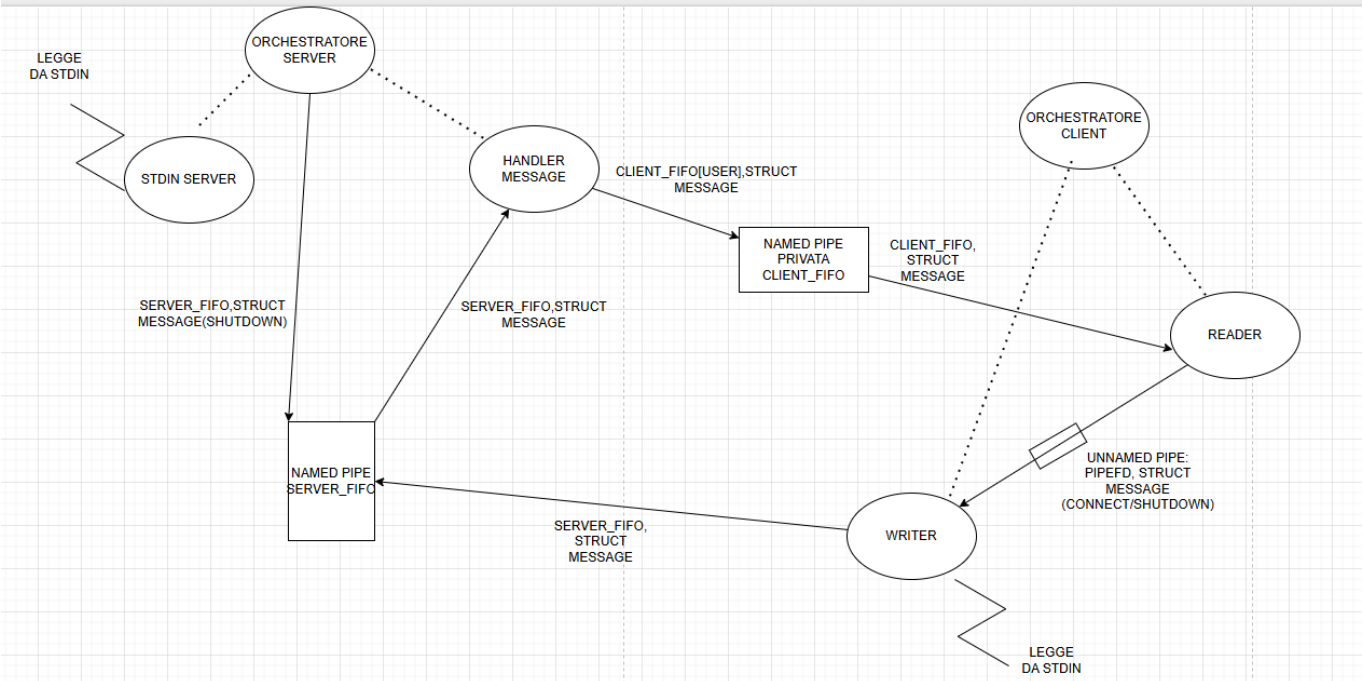
2.3 Comunicazione

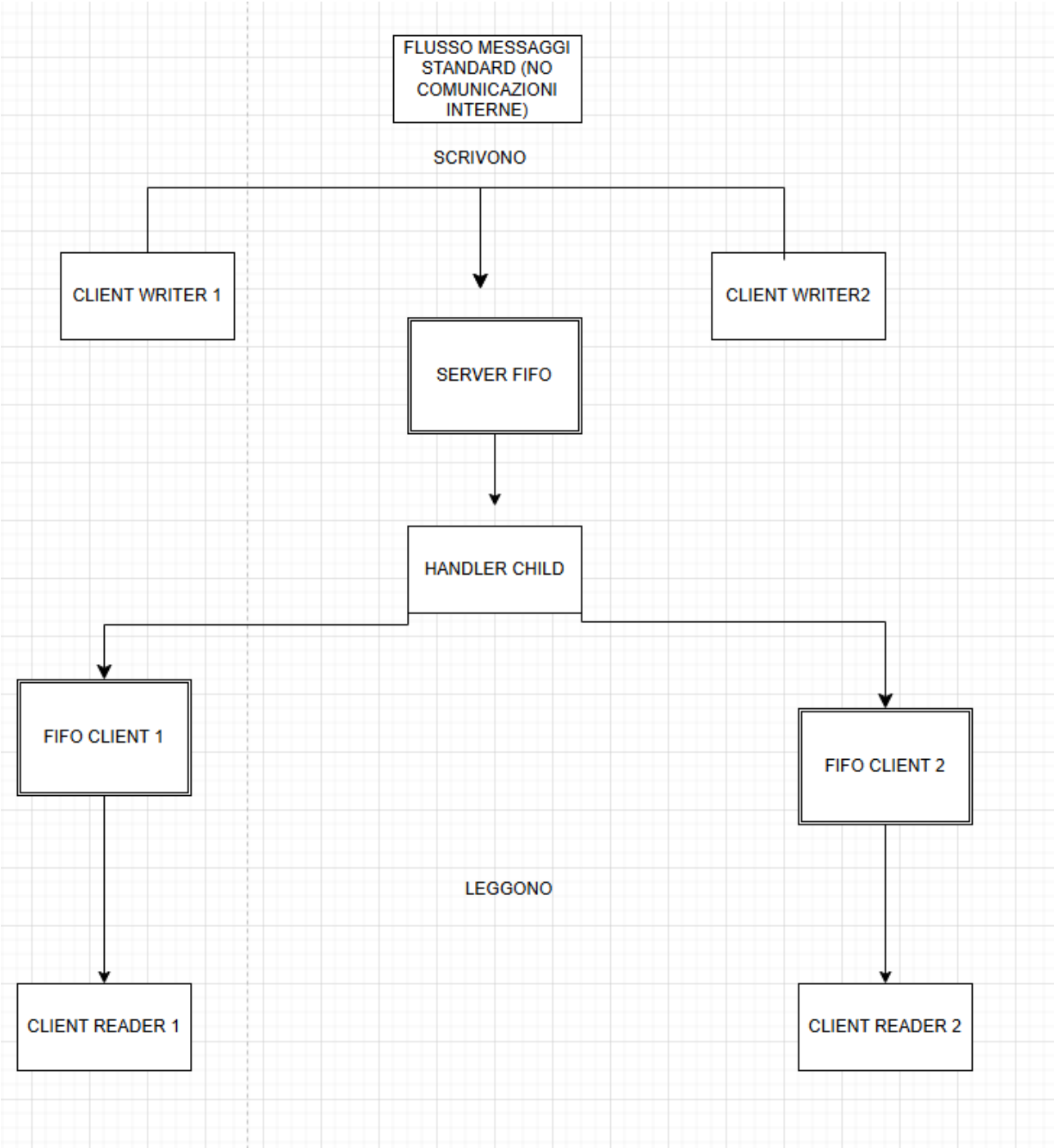
La comunicazione avviene attraverso:

- **FIFO del Server**: Tutti i client inviano messaggi al server tramite questa FIFO (handler legge da questa e smista i messaggi) (può essere creata prima dell'avvio del server con il comando mkfifo ma non è necessario, basta passare "server_fifo" (i client sanno il nome fifo per aprirla e scriverci)
- **FIFO Private Client**: Il server invia messaggi ai client tramite queste FIFO (create con l'esecuzione del programma client sui terminali)
- **Pipe interne**: Utilizzate per la comunicazione tra processi dello stesso client

2.4 Gestione dei processi

Diagramma dei processi:


LEGGE
DA STDIN
ORCHESTRATORE
SERVER
STDIN SERVER
HANDLER
MESSAGE
CLIENT_FIFO[USER],STRUCT
MESSAGE
ORCHESTRATORE
CLIENT
NAMED PIPE
PRIVATA
CLIENT_FIFO
CLIENT_FIFO,
STRUCT
MESSAGE
READER
SERVER_FIFO,STRUCT
MESSAGE(SHUTDOWN)
SERVER_FIFO,STRUCT
MESSAGE
NAMED PIPE
SERVER_FIFO
UNNAMED PIPE:
PIPEFD, STRUCT
MESSAGE
(CONNECT/SHUTDOWN)
SERVER_FIFO,
STRUCT
MESSAGE
WRITER
LEGGE
DA STDIN
PROGETTAZIONE CONSIDERANDO UN
SOLO CLIENT (AGGIUNGERE MODELLO
PARTE CLIENT PER OGNI CLIENT DA
AGGIUNGERE)



Flowchart e Pseudo codice:

flowchart server

1) Parent (Orchestratore)
start Orchestratore
inizializza utenti
(crea la fifo server)
crea figlio Handler child (
crea Shutdown
attende che il watcher legga “shutdown” da stdin e termini
invia shutdown alla fifo in modo che anche l’Handler lo riceva
attende che l’Handler termini dopo aver gestito lo shutdown
libera le risorse
end Orchestratore

2.Handler child
start Handler
loop :

in base al select capisce se leggere dalla fifo server i messaggi (nel caso li smista)
scrivendo sulle fifo private dei client
se legge lo shutdown libera le risorse e termina

end Handler

3.Shutdown child
start Shoutdown gestor
stampa ("Type 'shutdown' to terminate\n")
loop :
legge da stdin in maniera non bloccante
se utente digita shutdown
comunica sulla fifo server lo shutdown
libera le risorse
termina
end Watcher

15

```
function main():
  inizializza_array_utenti()
  crea_server_fifo()
  apri_server_fifo()

  // Crea handler dei messaggi
  pid_handler = fork()
  if pid_handler == 0:
    while true:
      attendi_messaggi_con_select()
      if messaggio_disponibile:
        leggi_messaggio()
        handle_message(messaggio)
    exit()

  // Crea figlio peri shutdown
  pid_shutdown = fork()
  if pid_shutdown == 0:
    while true:
      leggi_da_stdin()
      if input == "shutdown":
        break
    exit()

  // Processo padre
  wait(pid_shutdown)
  invia_messaggio_shutdown()
  wait(pid_handler)
  chiudi_risorse()
  return

function handle_message(msg):
  switch msg.type:
    case MSG_CONNECT:    handle_connect(msg)
    case MSG_DISCONNECT: handle_disconnect(msg)
    case MSG_PUBLIC:     handle_public(msg)
    case MSG_PRIVATE:    handle_private(msg)
    case MSG_CHANGECH:   handle_change_channel(msg)
    case MSG_SHUTDOWN:   shutdown_server()
```

pseudocodice server

11

---

flowchart client

1. Flowchart del Writer (scrittore)
Start
Invia MSG_CONNECT al server
Attende (read) il segnale “1” dal Reader (pipe)
Stampa il menu e il prompt >
Loop
Esegui select({stdin, pipe})
Se la pipe è pronta:
read(code)
Se code == 2 → break (esci dal loop)
Se stdin è pronto:
fgets(line)
Rimuovi \n (aggiusta(line))
Decidi il tipo di comando:
“/exit” → invia MSG_DISCONNECT → break
“/channel X” → invia MSG_CHANGECH con channel = X
“@user msg” → invia MSG_PRIVATE a user con text = msg
altrimenti → invia MSG_PUBLIC con text = line
Ristampa il prompt >
End (exit writer)

2.Flowchart del Reader (lettore)
Start
Apri la FIFO privata
Loop
leggo i messaggi dalla fifo privata read(fifo, &msg)
Se msg.type == MSG_CONNECT → break
Invia “1” al Writer (pipe) per conferma CONNECT
Loop
leggo i messaggi dalla fifo privata read(fifo, &msg)
Switch su msg.type:
MSG_PUBLIC: stampa "[channel][sender]: text"
MSG_PRIVATE: stampa "[priv][sender->you]: text"
MSG_JOIN/MSG_LEAVE: stampa "*** sender text"
MSG_ERROR: stampa " ERROR: text"
MSG_SHUTDOWN o MSG_DISCONNECT:
stampa "[server]: text"
invia “2” al Writer (pipe)
Exit (esci dal Reader)
End (exit reader)

14

```
function main(username, initial_channel):
    crea_fifo_privata(username)
    crea_pipe_interna()

    // Processo figlio reader
    pid_reader = fork()
    if pid_reader == 0:
        apri_fifo_privata()
        attendi_conferma_connessione()
        segnala_connessione_a_writer()

        while true:
            leggi_messaggio_da_fifo()
            if messaggio.type in [MSG_SHUTDOWN, MSG_DISCONNECT]:
                notifica_writer_di_uscire()
                break
            visualizza_messaggio()
        exit()

    // Processo figlio writer
    pid_writer = fork()
    if pid_writer == 0:
        invia_richiesta_connessione(username, initial_channel)
        attendi_conferma_da_reader()
        mostra_menu_comandi()

        while true:
            usa_select_per_monitorare(stdin, pipe_reader)
            if input_su_stdin:
                leggi_input()
                if input == "/exit":
                    invia_disconnect()
                    break
                else if input inizia con "/channel":
                    invia_cambio_canale()
                else if input inizia con "@":
                    invia_messaggio_privato()
                else:
                    invia_messaggio_pubblico()
            if segnale_su_pipe:
                leggi_segnale()
                if segnale == "exit":
                    break
        exit()

    // Processo padre
    wait(pid_reader)
    wait(pid_writer)
    chiudi_risorse()
    return
```

pseudocodice client

1

3. Motivazioni delle Scelte Progettuali, Limiti e Vantaggi

Motivazioni delle Scelte Progettuali:

- **Architettura client-server**: Garantisce separazione tra logica di comunicazione e interfaccia utente
- **FIFO multiple**: Una FIFO condivisa per il server e FIFO individuali per ogni client consentono comunicazioni efficienti
- **Process-based**: La scelta di utilizzare processi separati invece di thread semplifica la gestione della memoria condivisa
- **Processi separati per lettura/scrittura**: Separare le funzioni di lettura e scrittura permette comunicazioni bidirezionali senza blocchi

Vantaggi:

- **Flessibilità**: Separazione chiara tra client e server
- **Gestione asincrona**: I client possono ricevere messaggi mentre ne stanno scrivendo
- **Modularità**: Separazione delle responsabilità tra processi
- **Facilità di espansione**: Possibilità di aggiungere nuove funzionalità
- **Comunicazione multicanale**: Gli utenti possono comunicare in canali separati o privatamente

Limiti e Problemi:

- **Numero fisso di utenti**: Max 20 utenti simultanei (array statico)
- **Lunghezza fissa**: Fissata lunghezza massima di messaggi, nomi utenti e nome canale
- **Unicità utenti**: Non ci possono più utenti attivi con lo stesso nome
- **Persistenza**: Nessuna persistenza dei messaggi dopo la disconnessione
- **Gestione degli errori**: Limitata gestione dei casi di errore e uscita improvvisa
- **Sicurezza**: Nessuna crittografia o autenticazione avanzata
- **Gestione corretta accesso risorse**: Possibili problemi di sincronizzazione/deadlock nell'accesso alle FIFO e lettura stdin
- **Difficoltà incontrate**: Gestione corretta della terminazione dei processi e sincronizzazione tra processi

4. Soluzione in POSIX C del Problema

4.1 Strutture Dati Principali

```
// Enumerazione per i tipi di messaggi
typedef enum {
```

```
    MSG_PUBLIC=0,   // Messaggio pubblico  canale
    MSG_PRIVATE,    // Messaggio privato tra utenti
    MSG_CHANGECH,   // Richiesta cambio canale
    MSG_DISCONNECT, // Disconnessione utente
    MSG_CONNECT,    // Connessione utente
    MSG_SHUTDOWN,   // Spegnimento server
    MSG_ERROR,      // Messaggio  errore
    MSG_JOIN,       // Notifica ingresso  canale
    MSG_LEAVE       // Notifica uscita  canale
} MsgType;

// Struttura per i messaggi
typedef struct {
    MsgType type;
    char sender[MAX_NAME];      // Nome  mittente
    char recipient[MAX_NAME];   // Nome del destinatario (msg privati)
    char channel[MAX_NAME];     // Nome  canale
    char text[MAX_TEXT];        // Contenuto messaggio
} Message;

// Struttura per gestire gli utenti
typedef struct {
    int active;             // Stato utente (0=inattivo, 1=attivo)
    char name[MAX_NAME];        // Nome utente
    char channel[MAX_NAME];     // Canale attuale
    char fifo[MAX_NAME + 8];    // Nome della FIFO privata dell'utente
} User;
```

4.2 Utilizzo della select() e Gestione IO non bloccante

La funzione select è utilizzata sia nel client che nel server per:

1. **Nel client (writer)**: Monitorare contemporaneamente lo stdin e la pipe di comunicazione interna

```
fd_set rd;
FD_ZERO(&rd);               // Inizializza set di file descriptor
FD_SET(STDIN_FILENO, &rd);     // Aggiunge stdin al set
FD_SET(pipefd[0], &rd);        // Aggiunge pipe al set
int mx = STDIN_FILENO > pipefd[0] ? STDIN_FILENO : pipefd[0];
select(mx+1, &rd, NULL, NULL, NULL);  // Attesa eventi
```

2. **Nel server**: Monitorare FIFO del server per nuovi messaggi

```
fd_set rd;
FD_ZERO(&rd);
FD_SET(fd_server, &rd);
select(fd_server + 1, &rd, NULL, NULL, NULL);
```

Vantaggi uso di select():

- Permette di gestire più fonti di input senza bloccarsi
- Il client può ricevere messaggi mentre attende input dall'utente

5. Istruzioni per Eseguire e Testare il Sistema

5.1 Compilazione

gcc –o server.exe server.c

gcc –o client.exe client.c

//PUOI CREARE IL FILE FIFO PRIMA

mkfifo server_fifo

5.2 Esecuzione del Server (su un terminale)

Prima avvio il server

./server.exe server_fifo

devo passare il nome della fifo server-> server_fifo

5.3 Esecuzione dei Client (un terminale per ogni client)

Poi avvio i client

./client.exe username channel_name

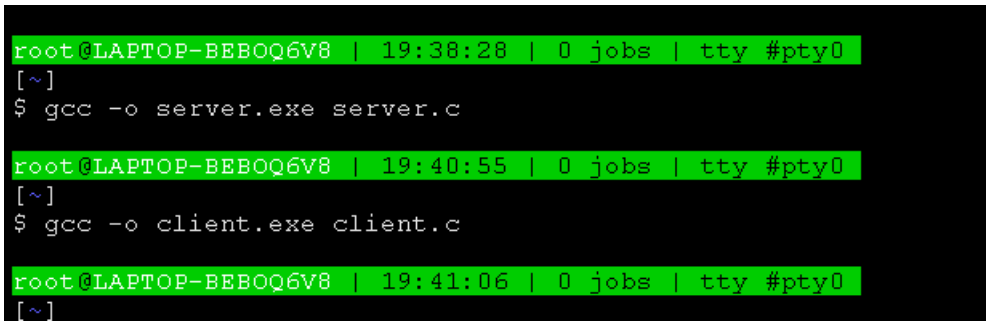
5.4 Comandi disponibili nei client

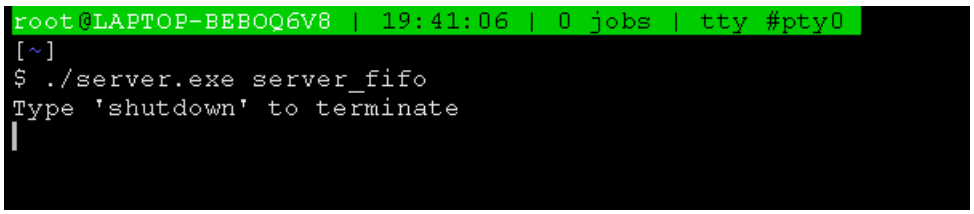
- /exit - Disconnette il client
- /channel nome_canale - Cambia canale al canale specificato
- @utente messaggio - Invia messaggio privato
- testo normale - Invia messaggio pubblico al canale attuale

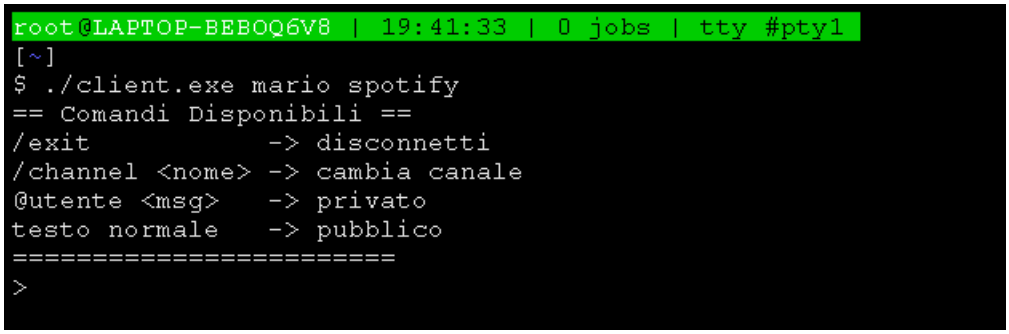
5.5 Chiusura del Server

Digitando shutdown nell'input del server, il sistema si chiuderà ordinatamente (comunicando ai client di terminare).

6. Screenshots







```
root@LAPTOP-BEBOQ6V8 | 19:41:35 | 0 jobs | tty #pty2
[~]
$ ./client.exe gloria spotify
== Comandi Disponibili ==
/exit           -> disconnetti
/channel <nome> -> cambia canale
@utente <msg>   -> privato
testo normale   -> pubblico
========================
>
```

```
root@LAPTOP-BEBOQ6V8 | 19:41:33 | 0 jobs | tty #pty1
[~]
$ ./client.exe mario spotify
== Comandi Disponibili ==
/exit           -> disconnetti
/channel <nome> -> cambia canale
@utente <msg>   -> privato
testo normale   -> pubblico
========================
*** gloria joined the channel.
> ciao
> @gloria come va
>
```

```
root@LAPTOP-BEBOQ6V8 | 19:44:20 | 0 jobs | tty #pty3
[~]
$ ./client.exe debo giochi
== Comandi Disponibili ==
/exit           -> disconnetti
/channel <nome> -> cambia canale
@utente <msg>   -> privato
testo normale   -> pubblico
========================
> ciaooo
>
```

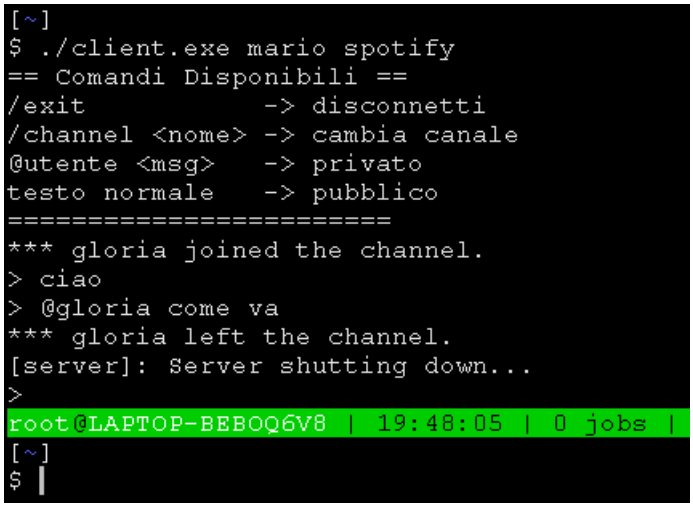
```
[~]
$ ./client.exe debo giochi
== Comandi Disponibili ==
/exit           -> disconnetti
/channel <nome> -> cambia canale
@utente <msg>   -> privato
testo normale   -> pubblico
========================
> ciaooo
> ciao
> /exit
[server]: You have been disconnected.
>
```

```
root@LAPTOP-BEBOQ6V8 | 19:41:35 | 0 jobs | tty #pty2
[~]
$ ./client.exe gloria spotify
== Comandi Disponibili ==
/exit           -> disconnetti
/channel <nome> -> cambia canale
@utente <msg>   -> privato
testo normale   -> pubblico
========================
[spotify][mario]: ciao
[priv][mario->you]: come va
> /channel giochi
>
```

```
root@LAPTOP-BEBOQ6V8 | 19:41:33 | 0 jobs | tty
[~]
$ ./client.exe mario spotify
== Comandi Disponibili ==
/exit           -> disconnetti
/channel <nome> -> cambia canale
@utente <msg>   -> privato
testo normale   -> pubblico
========================
*** gloria joined the channel.
> ciao
> @gloria come va
*** gloria left the channel.
>
```

```
root@LAPTOP-BEBOQ6V8 | 19:41:06 | 0 jobs | t
[~]
$ ./server.exe server_fifo
Type 'shutdown' to terminate
shutdown
[Server] Shutdown complete.

root@LAPTOP-BEBOQ6V8 | 19:48:05 | 0 jobs | t
```



GRAZIE PER L'ATTENZIONE