PROGETTAZIONE DISCORD – TEPSIT

Morelli Davide 4CI

Progetto:

Creare un sistema di chat minimalista che simuli le funzionalità base di Discord in C, usando le named pipe.

Per la creazione del codice ho deciso di optare per un singolo canale che va ad ospitare tutti i messaggi. Per avere una struttura più semplice da creare.

Nel pratico:

/tmp/General funge da canale condiviso, si tratta del server generale per tutti i messaggi di tutti gli utenti. Il server gestisce la FIFO, i client leggono o scrivono su di essa. Ogni client usa fork() per ascoltare (figlio) e inviare (padre) contemporaneamente.

Per testare:

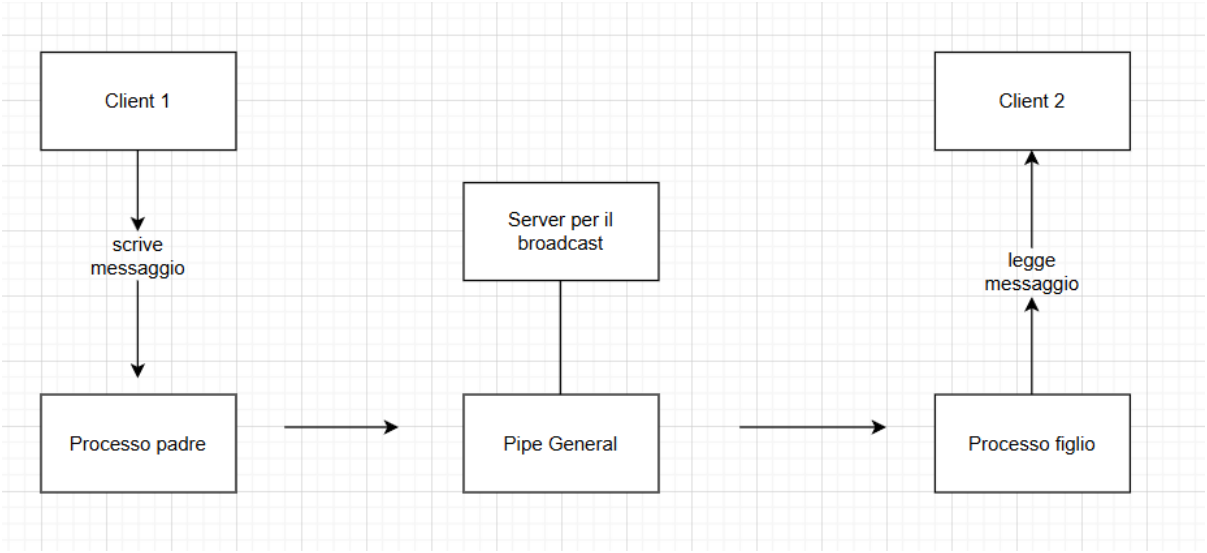
Alla fine, ho implementato 1 terminale server e 3 terminali client. Ho poi testato che i messaggi venissero ricevuti da tutti i client e che si verificasse un'uscita pulita con exit, anche se purtroppo i figli rimangono attivi dopo la chiusura. Dovrebbero poi essere terminati manualmente.

Funzioni POSIX Usate:

Server: mkfifo(), open(), read(), write().

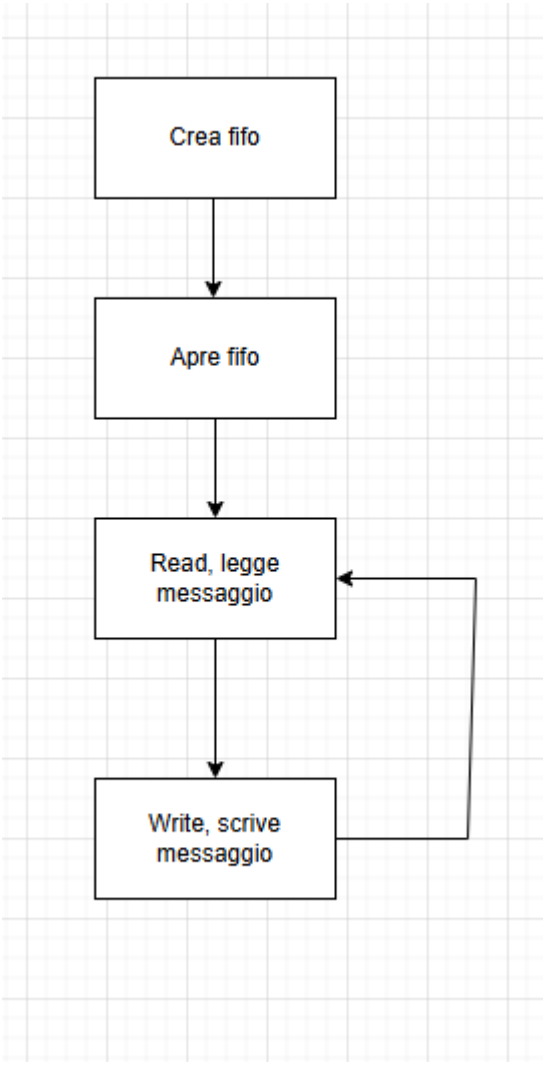
Client: fork(), open(), read(), write().

Infine, questo è ciò che accade quando viene scritto un messaggio:

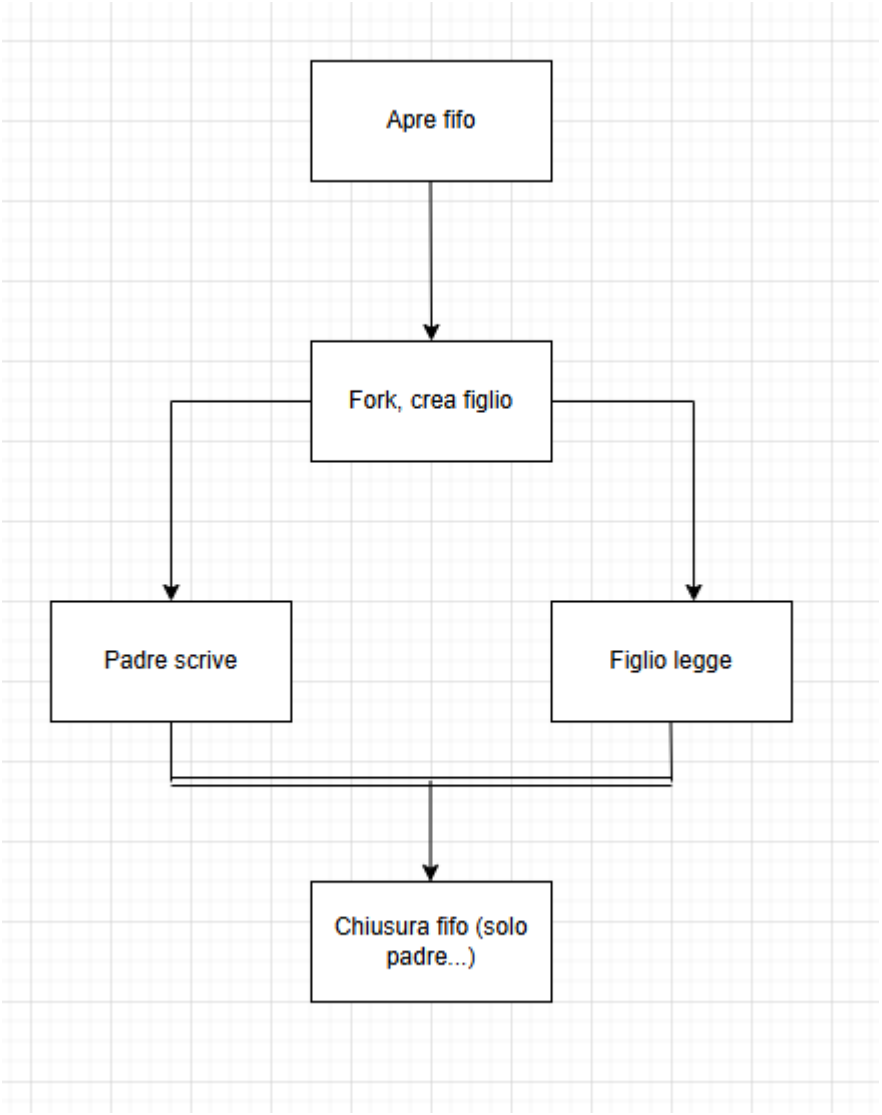


Il processo può accadere al contrario, è solo un esempio.

Il funzionamento individuale del codice server, invece, è il seguente:



Quello del codice utente è invece questo:



Conclusione:

Con l'esecuzione del programma del server, verrà visualizzato il seguente output dal sistema:

```
root@LAPTOP-7RLE9F1U | 22:35:59 | 0 jobs | tty #pty2
[~]
$ cd ~/discord_project   # This is your project folder
ls                     # Verify server.c and user.c are listed
server.c  server.exe

root@LAPTOP-7RLE9F1U | 22:36:56 | 0 jobs | tty #pty2
[~/discord_project]
$ touch server.c

root@LAPTOP-7RLE9F1U | 22:37:09 | 0 jobs | tty #pty2
[~/discord_project]
$ notepad server.c

root@LAPTOP-7RLE9F1U | 22:37:24 | 0 jobs | tty #pty2
[~/discord_project]
$ gcc server.c -o server

root@LAPTOP-7RLE9F1U | 22:37:34 | 0 jobs | tty #pty2
[~/discord_project]
$ ./server
SERVER RUNNING. BROADCASTING TO /tmp/General...
```

Con l'esecuzione della parte utente, invece, sarà questo l'output:

```
root@LAPTOP-7RLE9F1U | 23:04:24 | 0 jobs | tty #pty0
[~]
$ cd ~/discord_project   # This is your project folder
ls                     # Verify server.c and user.c are listed
server.c  server.exe  user.c

root@LAPTOP-7RLE9F1U | 23:05:01 | 0 jobs | tty #pty0
[~/discord_project]
$ touch user.c

root@LAPTOP-7RLE9F1U | 23:05:08 | 0 jobs | tty #pty0
[~/discord_project]
$ notepad user.c

root@LAPTOP-7RLE9F1U | 23:05:19 | 0 jobs | tty #pty0
[~/discord_project]
$ gcc user.c -o user

root@LAPTOP-7RLE9F1U | 23:05:42 | 0 jobs | tty #pty0
[~/discord_project]
$ ./user
ciao
RECEIVED: ciao
ciao ciao ciao
RECEIVED: ciao ciao ciao
```

Ogni cosa inserita dall'utente verrà messa come ricevuta, anche i suoi stessi messaggi,

che verranno visualizzati insieme a tutti gli altri messaggi di tutti gli altri utenti su discord (nel nostro caso altri 2 utenti).

Errori e complicazioni:

-Ho riscontrato alcuni problemi nella trasmissione dei messaggi sul server. Non sono riuscito a capire la causa del problema.

-Non vengono automaticamente eliminati tutti i processi correttamente nel codice, è quindi necessario andare a terminare i processi figli manualmente per l'utente.

-Non ci sono precauzioni in caso di errore di write o read, per rendere il codice più corto e semplice, ma sacrificando dell'ottimizzazione.

-Non ci sono nomi utente o identificativi di qualsiasi tipo per i vari utenti del server, i messaggi sono semplicemente tutti inseriti anonimamente.

-C'è un solo canale (ovvero General), rendeva il progetto più facile per me.

-Se un messaggio immesso supera i 256 caratteri, il sistema si rompe.

-Nessuna password o metodo di accesso, l'utente semplicemente nasce.