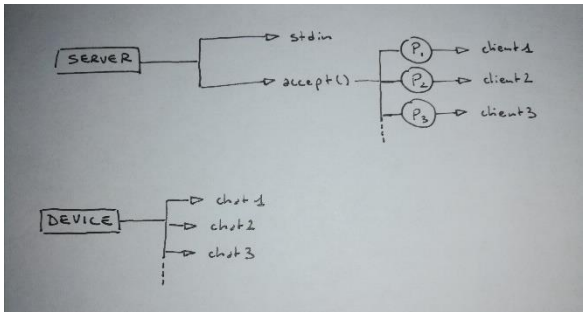
PROGETTO di RETI INFORMATICHE

a cura di Luca Davini a.a2021/2022

IMPLEMENTAZIONE CLIENT E SERVER

Lato Client ho deciso di utilizzare un singolo processo che gestisca tutte le chat contemporaneamente tramite la *select()*.
Ciò è stato possibile considerando che un client partecipa attivamente a una singola chat alla volta, mantenendo tutte le altre in background con la sola possibilità di ricezione di messaggi, ottima condizione per l'utilizzo della *select()* per controlli in lettura.

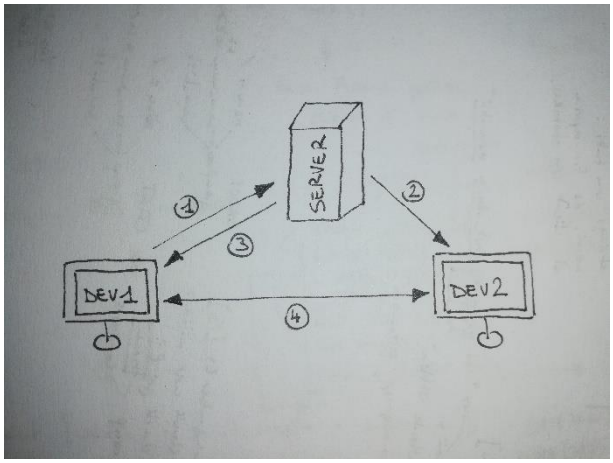
Lato Server ho utilizzato una soluzione ibrida: un processo gestisce tramite *select()* gli input da tastiera e le richieste di connessione dei client. Per ogni nuova richiesta viene generato un processo figlio che si dedichi unicamente al nuovo client connesso.
Con l'utilizzo di un processo dedicato al singolo client si evita che un utente possa rallentare il servizio, mentre con l'utilizzo della *select()* si evita di costruire una gerarchia di processi difficile da controllare.



COSTRUZIONE DI UNA CHAT

Il meccanismo di creazione di una chat prevede che un utente *dev1*, per aprire una nuova chat con *dev2*, contatti il server che si occuperà di verificare se questo sia online o meno:

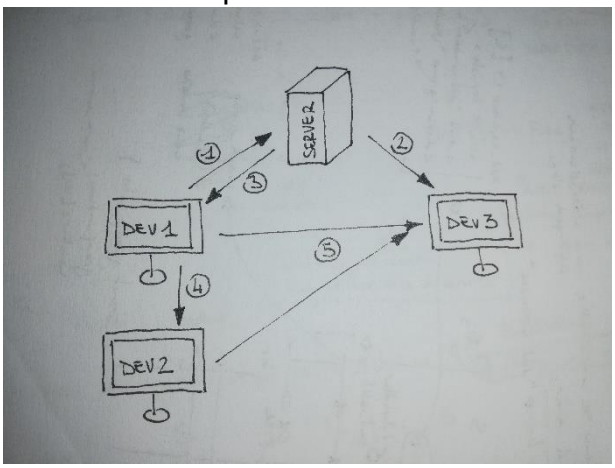
- se *dev2* risulta offline, viene automaticamente aperta una *chat offline* con il server;
- se *dev2* risulta online, visualizza la richiesta "*dev1 vuole chattare. Aprire la chat?*" a cui dovrà rispondere con "*S*" o "*N*".
  Ho ritenuto positivo l'utilizzo di questo meccanismo che permette all'utente di scegliere come proseguire. Questo blocca però il servizio in attesa della risposta e presenta un punto debole: se, al momento dell'arrivo della richiesta, *dev2* stava scrivendo, il buffer di *stdin* risulterà sporco, compromettendo la risposta alla domanda (*vedi README.txt*).



COSTRUZIONE DI UNA CHAT DI GRUPPO

Una chat di gruppo può essere aperta soltanto se tutti i partecipanti sono online. Questa risulta attiva se tutti i partecipanti la hanno aperta (*chat corrente*). Ciò implica che quando un utente si disconnette o esce dalla chat, questa viene chiusa per tutti.

Considerando una chat attiva tra *dev1* e *dev2*, l'aggiunta di un utente alla chat prevede di contattare il server che sceglie un nome per il gruppo e notifica la richiesta all'utente *dev3*, il quale entra in un ciclo in attesa che gli altri utenti gli richiedano di stabilire una connessione. Il server risponde quindi all'utente richiedente con le informazioni sull'utente da contattare che sono poi distribuite agli altri membri della chat per permettergli stabilire delle connessioni con *dev3*.

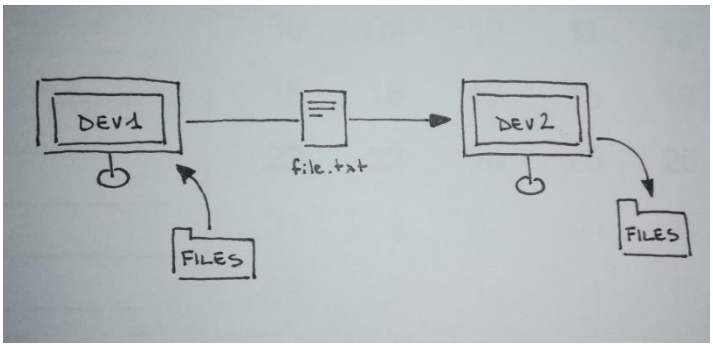


Ho ritenuto più comodo far ricevere a *dev3* le richieste di connessione, poiché per fargliele effettuare avrei dovuto raccogliere i dati di tutti i partecipanti alla chat di partenza. In questo modo invece devo condividere solamente le informazioni su *dev3* tra gli utenti della chat originaria, riducendo il quantitativo di dati da gestire.

Una volta stabilite tutte le connessioni necessarie, il gruppo viene aperto, diventando la *chat corrente* per tutti i partecipanti, senza possibilità di decisione. Ciò è stato scelto per comodità, limitando la possibilità di scelta degli utenti. Negativo è il fatto che se un utente sta partecipando ad un gruppo e viene invitato in un altro, all'apertura del secondo l'utente uscirà automaticamente dal primo, forzando la chiusura anche per tutti gli altri partecipanti.

PROTOCOLLO DI CONDIVISIONE FILE

La condivisione di un file inizia con l'invio del messaggio "*share file_name*" in una chat. L'utente richiedente recupera il file dalla propria cartella "*FILES*" e invia una notifica di condivisione file ("*SHARE_FILE*") ai partecipanti della chat, permettendogli di prepararsi alla ricezione. Questi ultimi, quando ricevono il messaggio "*SHARE_FILE*", controllano se esiste già nella propria cartella "*FILES*" un file omonimo, nel qual caso procedono aggiungendo al nome del file da ricevere un identificativo generato dalla funzione "*time(NULL)*".



L'utilizzo della cartella "*FILES*" permette di semplificare la ricerca e il recupero del file, mentre la ridenominazione tramite timestamp permette di garantire univocità, senza la necessità di memorizzare un indice, anche se rende meno leggibile il nome del file.

RIEPILOGO di PREGI e DIFETTI

AFFIDABILITÀ: grazie all'utilizzo del protocollo TCP, ho la garanzia di affidabilità del trasporto dei dati;

SCALABILITÀ: l'utilizzo di strutture per lo scambio di messaggi permette di velocizzare eventuali processi di modifica dei formati;

DISTRIBUZIONE COMPLESSITÀ: gestendo i chat log lato client riduco la complessità lato server.

TEMPESTIVITÀ: l'utilizzo di TCP non permette di contare sulla tempestività dell'invio e ricezione dei dati;

COMPLESSITÀ: l'utilizzo di molte strutture richiede una buona conoscenza del progetto e dei vari formati utilizzati;

SINGLE POINT OF FAILURE: la gestione centralizzata di informazioni fondamentali per il servizio lo rende vulnerabile ai guasti.