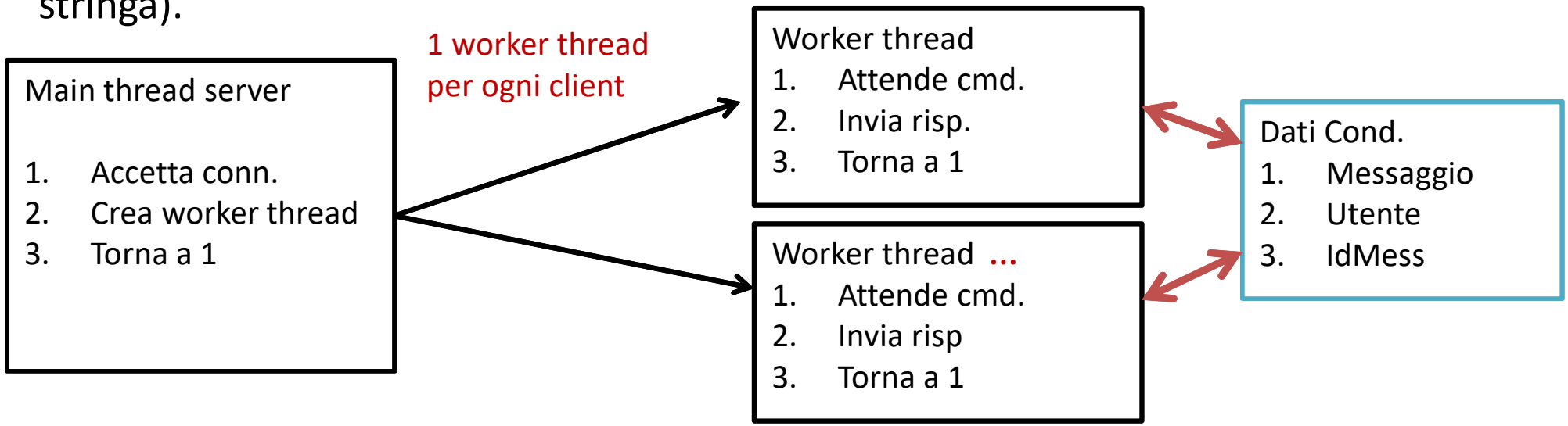
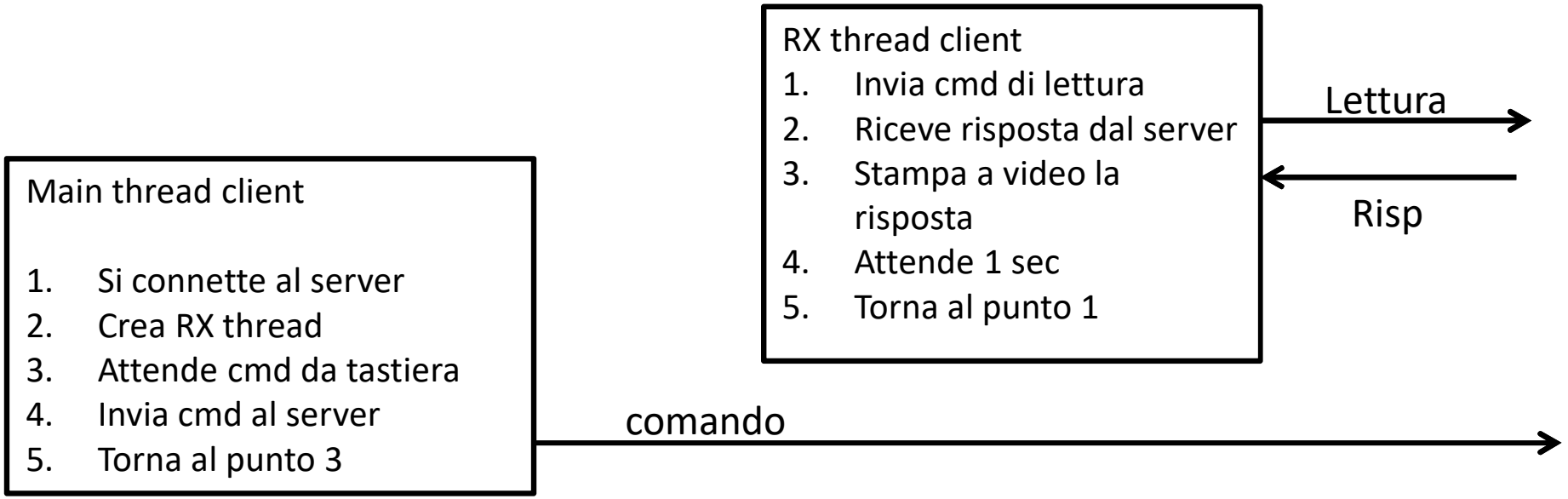
Realizzare un sistema di messaggistica remota in cui un server gestisce lo scambio di messaggi tra molteplici client remoti (tutti i client devono essere uguali).
Il programma prevede la realizzazione di un server e di un client.

- Il server **attende la connessione da parte di più client**. Una volta stabilita la connessione con un client, **il server avvia un thread** (chiamato worker_thread) che si occuperà di gestire la comunicazione tra il server e quel client.
- Il worker_thread ciclicamente, fino a quando non viene chiusa la connessione, **attende un comando** (in formato stringa) dal client **e invia una risposta** (in formato stringa).



Il client, dopo aver stabilito la connessione, crea un thread secondario.

- Il thread secondario periodicamente e ad intervalli di un secondo invia un comando di lettura per richiedere al server l'ultimo messaggio inviato al server, ne attende la risposta e la stampa a video.
- Il thread principale del client ciclicamente richiede all'utente di inserire un comando da tastiera. Non appena è inserito un comando, lo invia al server.



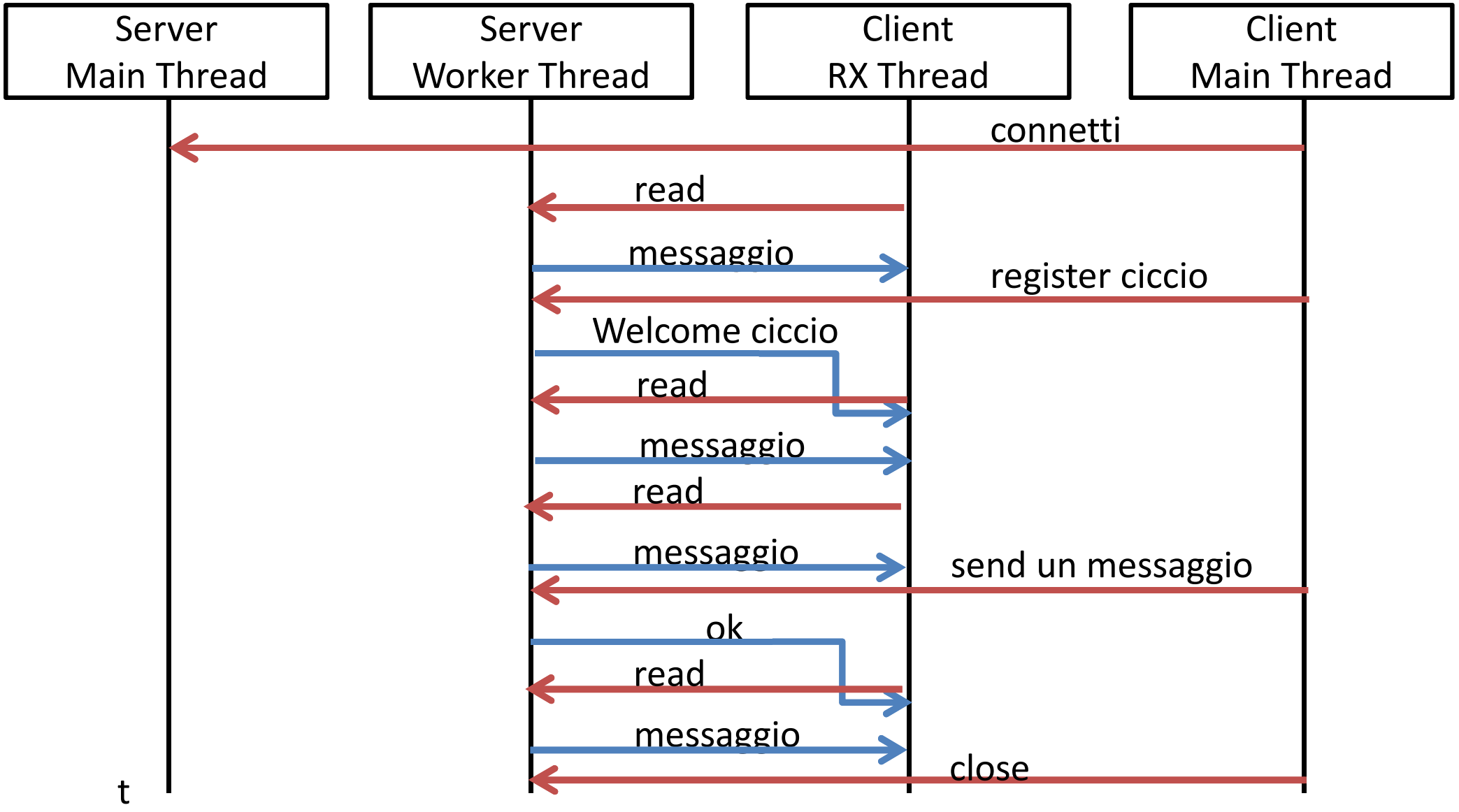
Elenco comandi a cui il server deve rispondere:

register <nome> - Con questo comando il server assegna un nome utente alla connessione col client. Il server risponde col messaggio: ‘Welcome <nome>’

send <messaggio> - Con questo comando il server memorizza in una variabile condivisa tra i thread il messaggio inviato e il nome utente. Il server risponde con ‘ok’

read - Con questo messaggio si richiede al server di inviare in risposta l’ultimo messaggio trasmesso. Il server risponde con una stringa nel formato: <idmessaggio> <mittente>: <messaggio>

close - Con questo messaggio il server chiude la connessione col client.


Server
Main Thread
Server
Worker Thread
Client
RX Thread
Client
Main Thread
connetti
read
messaggio
register ciccio
Welcome ciccio
read
messaggio
read
messaggio
send un messaggio
ok
read
messaggio
close
t

Motivazioni per cui serve un server concorrente:

Più client sono connessi al server allo stesso tempo, inoltre non si ha sincronizzazione tra le richieste che vengono inviate dai vari client.

Ossia, non si sa quali client invii un messaggio per primo.

Alcune possibili metodologie di implementazione:

- Utilizzare la fork() per creare dei processi figli che gestiscano le connessioni con i client. Lo svantaggio, nel caso in esame, è che si devono prevedere opportuni meccanismi di comunicazione tra i processi del server.

- Utilizzare i thread nel server per gestire le connessioni con i client. Nell'esempio specifico poiché lo spazio degli indirizzi è condiviso tra i thread è semplice farli comunicare tra loro.