Assignment #2: Relazione

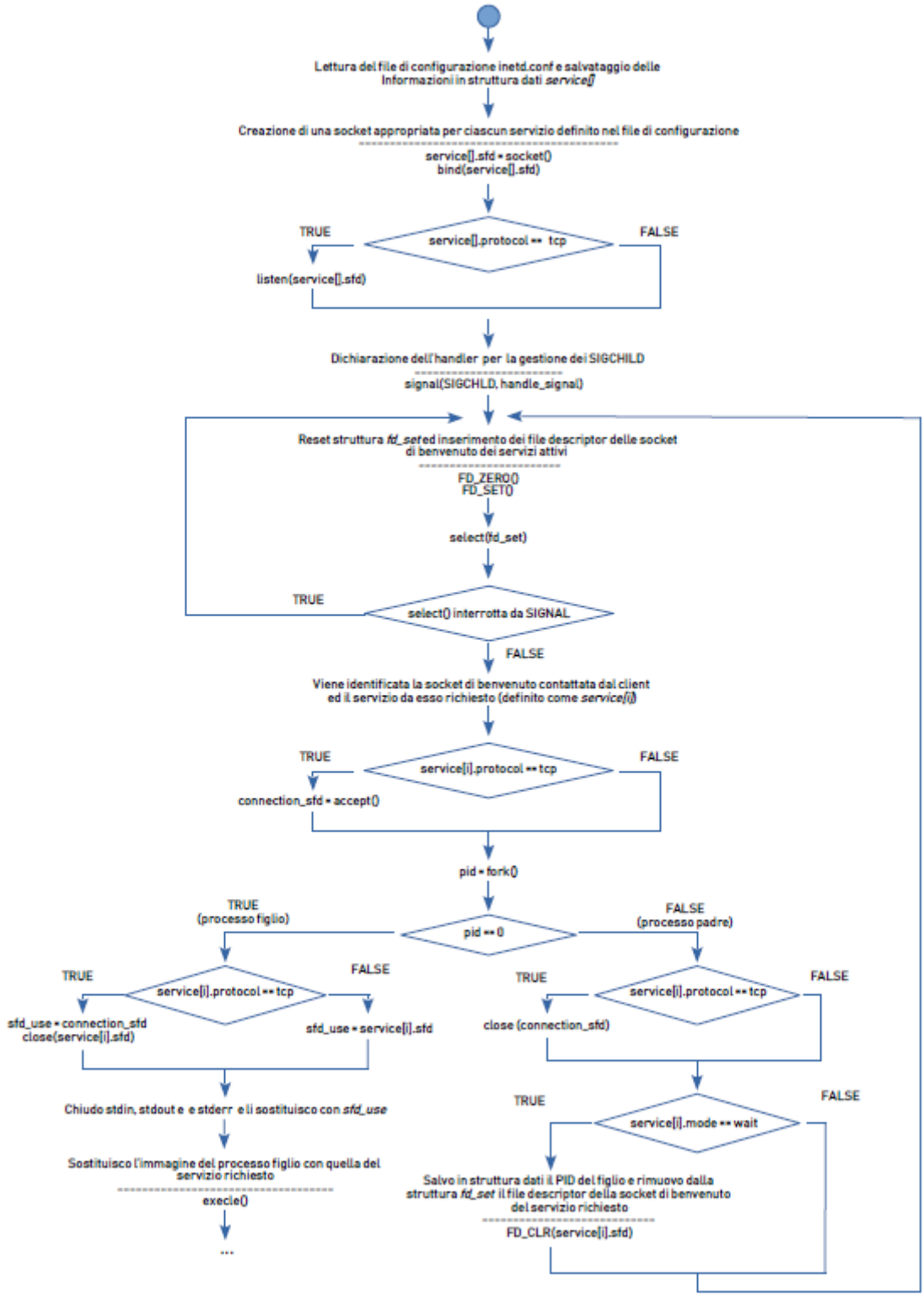
PROGRAMMAZIONE DI RETI

BAIARDI MARTINA
LOMBARDINI ALESSANDRO

TASK 1

Il superserver crea una socket per ogni servizio che mette a disposizione e le associa, tramite delle chiamate alla funzione *bind(),* alle porte stabilite all'interno di inetd.conf. Se il servizio è TCP, viene invocata la funzione *listen()*, che pone in ascolto la socket creata di eventuali richieste di connessione. Viene infine invocata la funzione *select()*, che permette di monitorare contemporaneamente più socket in attesa che queste vengano contattate da un client.

Giunta una nuova richiesta, se la connessione è TCP viene invocata la *accept()* che apre una socket di connessione con il client richiedente. Viene effettuata una *fork()*, cioè viene copiata l'immagine del processo del superserver in un nuovo processo, chiamato processo figlio, che viene poi opportunamente sostituito con l'eseguibile del server richiesto per il servizio, attraverso la funzione *execle()*. Il processo padre, nonché il superserver, torna in ascolto di nuove richieste tramite una nuova chiamata alla funzione *select()*. Non viene qui mostrata la gestione delle signal.



TASK 2

Implementazione del Super-Server

Il superserver, per prima cosa, effettua una lettura del file di configurazione *inetd.conf* e salva le informazioni contenute al suo interno dentro una struttura dati preventivamente creata. Vengono memorizzati, per ogni tipologia di servizio:

- *Protocollo di trasporto*, TCP o UDP
- *Tipo di servizio*, wait o nowait
- *Porta del servizio*, porta contattata dai client per richiedere il servizio
- *Indirizzo completo del servizio*, indirizzo dell'eseguibile che offre il servizio
- *Nome del servizio*, nome del servizio (equivalente al nome dell'applicativo)
- *Socket File Descriptor*, socket file descriptor al quale è associato il servizio
- *Process Identifier*, codice univoco del processo figlio creato per soddisfare una richiesta del servizio. Questo parametro serve per gestire i servizi di tipo *wait*.

A questo punto il superserver procede alla creazione delle socket di benvenuto che devono essere associate ai diversi servizi disponibili. Per ciascun servizio:

1. Viene invocata la funzione ***socket()***
    - Se la socket è richiesta di tipo TCP, la funzione conterrà i parametri: AF_INET (Famiglia di indirizzi IPv4), SOCK_STREAM (Socket con connessione), IPPROTO_TCP (Protocollo TCP).
    - Se la socket è richiesta di tipo UDP, la funzione conterrà i parametri: AF_INET (Famiglia di indirizzi IPv4), SOCK_DGRAM (Socket senza connessione), IPPROTO_UDP (Protocollo UDP)
2. Viene chiamata la funzione ***bind()***
3. Se la socket generata è di tipo TCP viene invocata la funzione ***listen()***, che consente di porre una socket in ascolto di richieste di connessione.

Per gestire la terminazione dei processi figli del superserver, viene impostata la procedura ***handle_signal()*** tramite la funzione ***signal()***.

Il superserver entra in un loop senza fine all'interno del quale si occupa di gestire le richieste dei client sulle socket precedentemente create; questa operazione viene gestita attraverso la funzione ***select()***. Giunta una richiesta, dopo aver identificato il servizio richiesto, si controlla se sia di tipo TCP, in tal caso viene invocata la funzione ***accept()*** che crea una nuova socket su cui verranno trasmessi i dati della comunicazione con quello specifico client.

A questo punto viene invocata la ***fork()***, che crea un processo figlio con la stessa immagine di quello corrente. Al processo figlio vengono chiusi i file descriptor 0,1 e 2, che corrispondono rispettivamente ai file descriptor di input, output ed error e vengono tutti sostituiti con la socket che riceverà i messaggi: quella di benvenuto per servizi UDP e quella di connessione per servizi TCP. Dopodiché viene sostituita l'immagine del processo figlio con quella del processo server utile per soddisfare il servizio richiesto. Il processo padre ha il compito di rimuovere la socket di benvenuto del servizio aperto dalla lista fd_set fornita alla select, solo nel caso in cui questo sia di tipo wait e si occupa di chiudere la socket di connessione che è stata eventualmente creata in caso di creazione di un servizio tcp.

Infine, per gestire i segnali SIGCHLD inviati dai processi figli quando terminano, è stata disposta la procedura *handle_signal()* che controlla che il segnale arrivato sia effettivamente un SIGCHLD. Se il processo terminato era un servizio di tipo wait la socket viene riaggiunta alla lista fd_set; questo consentirà alla *select()* di controllarla per nuove richieste di servizi.

Istruzioni utilizzate per la compilazione

Il codice è stato compilato e linkato attraverso il supporto di un Makefile, qui riportato:

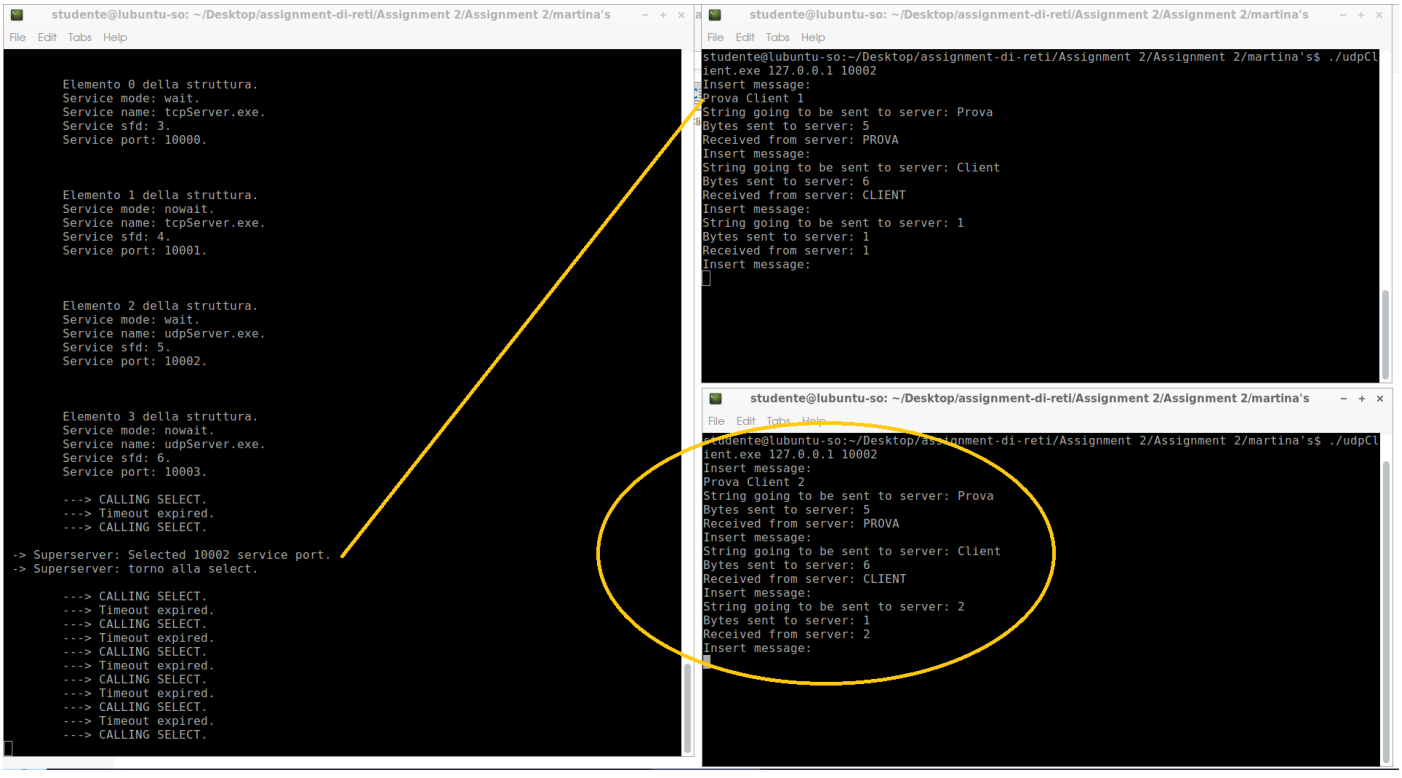
```
all:            superserver.exe udpClient.exe udpServer.exe tcpClient.exe tcpServer.exe

superserver.exe:    superserver.o
                    gcc ${CFLAGS} -o superserver.exe superserver.o

superserver.o:      superserver.c
                    gcc -c ${CFLAGS} superserver.c

udpClient.exe:      udpClient.o
                    gcc ${CFLAGS} -o udpClient.exe udpClient.o

udpClient.o:        udpClient.c
                    gcc -c ${CFLAGS} udpClient.c

udpServer.exe:      udpServer.o
                    gcc ${CFLAGS} -o udpServer.exe udpServer.o

udpServer.o:        udpServer.c
```

```
                gcc -c ${CFLAGS} udpServer.c
tcpClient.exe:  tcpClient.o
                gcc ${CFLAGS} -o tcpClient.exe tcpClient.o
tcpClient.o:    tcpClient.c
                gcc -c ${CFLAGS} tcpClient.c
tcpServer.exe:  tcpServer.o
                gcc ${CFLAGS} -o tcpServer.exe tcpServer.o
tcpServer.o:    tcpServer.c
                gcc -c ${CFLAGS} tcpServer.c
.PHONY:         clean
clean:
                -rm -f *.exe  *.o
```

UdpServer behavior in wait mode

Quando il superserver riceve una richiesta da parte di un client UDP per un servizio in wait mode, esso si occupa di eseguire un udpServer per gestirne la richiesta. Nel momento in cui sopraggiunge un nuovo client a richiedere lo stesso servizio, il superserver non accetta richieste in quanto in wait mode, dunque non vengono eseguiti ulteriori server. Il server precedentemente attivato è però attivo sempre sulla stessa porta, quindi risponde alla richiesta del nuovo client.

Abbiamo effettuato con il client A la richiesta "Prova Client 1", ottenendo l'esecuzione di un processo server che risponde correttamente al messaggio e rimane in attesa di ulteriori messaggi. E' stato poi utilizzato un secondo client (B) per effettuare una seconda richiesta, "Prova Client 2", al quale però il superserver, rispettando i vincoli della wait mode, non ha allocato alcun server. Il client B ha ottenuto ugualmente risposta in quanto il server dedicato ad A si è occupato di tale richiesta.

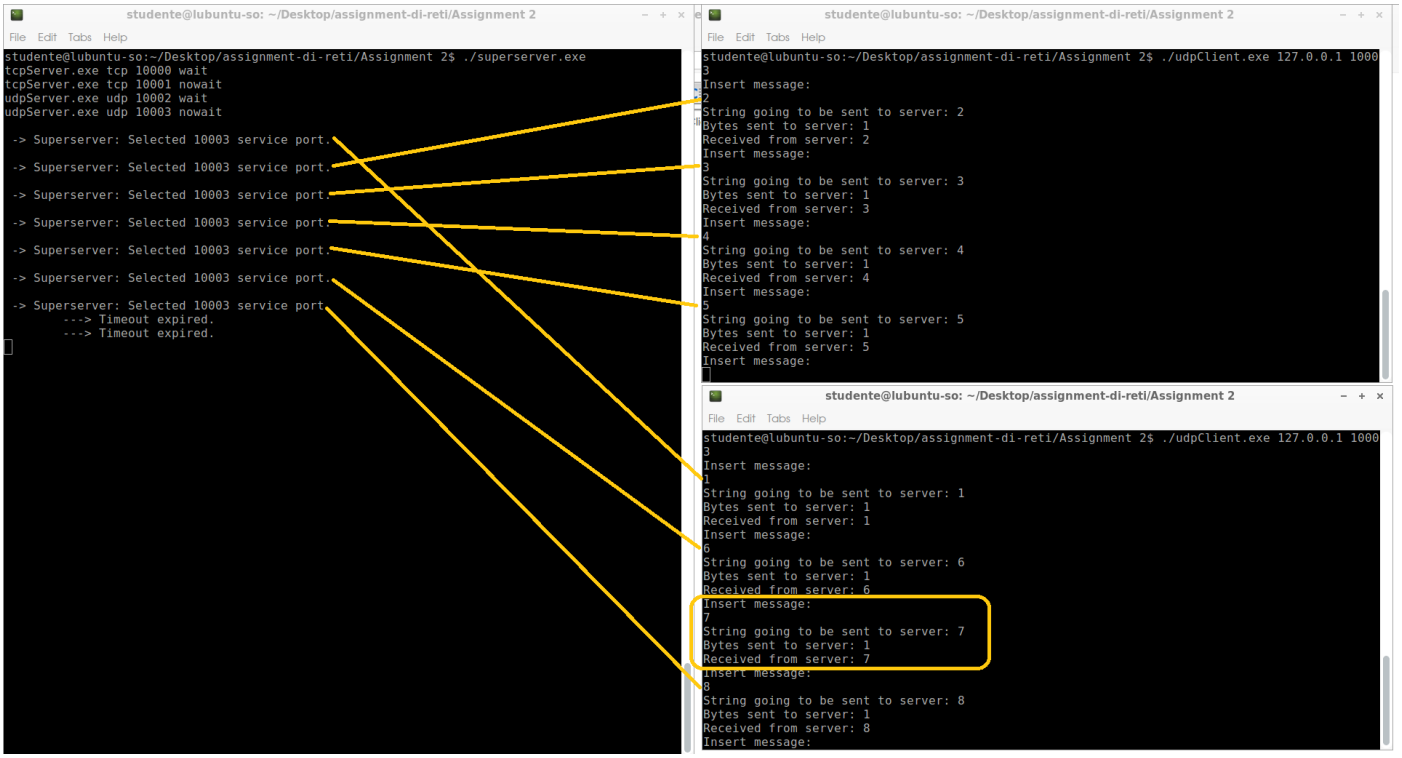


UdpServer behavior in no-wait mode

Quando il superserver riceve una richiesta da parte di un client UDP per un servizio in no-wait mode genera un processo udpServer, che si occuperà di rispondere alla richiesta del client. Siccome il servizio richiesto è di tipo no-wait, il superserver non toglie mai l'ascolto di nuove richieste di tale servizio, creando così una situazione di "concorrenza" tra superserver e i server udp precedentemente creati, in quanto tutti in attesa di richieste sulla stessa porta. Quando si inviano dei messaggi da un client, non è possibile prevedere chi tra superserver e udpServer eseguiti risponderà alla richiesta. Nello screen riportato, si è cercato di evidenziare questo fenomeno: se il messaggio viene ricevuto prima dal superserver, viene

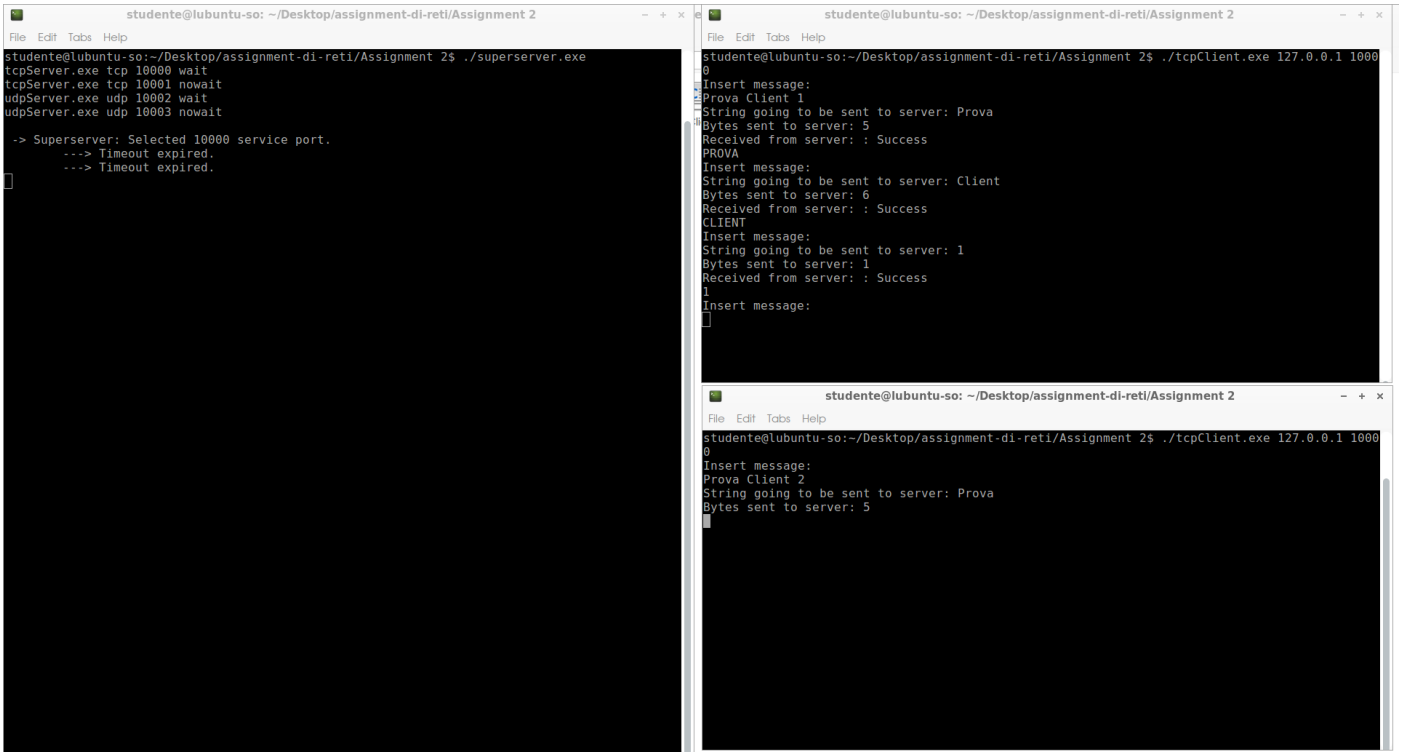
aggiunto un nuovo server sulla stessa service-port, che poi risponde alla richiesta; se invece il messaggio viene ricevuto prima da un udpServer, quest'ultimo risponde direttamente al client.

Per evidenziare meglio questo fenomeno, è stata inserita una *usleep()* al processo padre subito prima di tornare alla select, la quale permette di effettuare una attesa in micro-secondi, dei quali ne abbiamo inseriti 5. Tale scelta è stata valutata perché altrimenti non sarebbe stato possibile controllare il flusso di pacchetti inviati dal client.

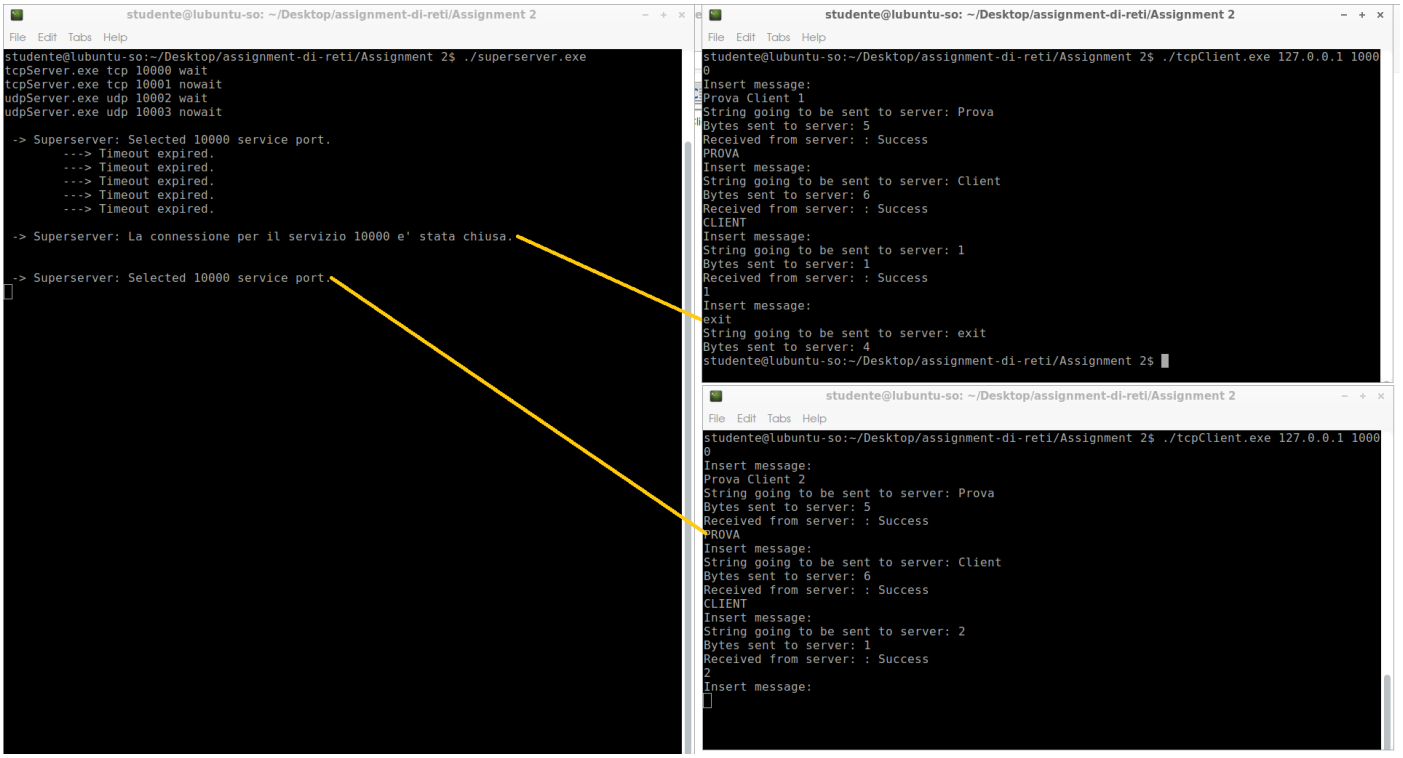


TcpServer behavior in wait mode

Quando il superserver riceve una richiesta da parte di un client TCP per un servizio in wait mode, alloca un server TCP il cui scopo è rispondere ai messaggi di quello specifico client. Se un nuovo client effettua una richiesta, non riceverà alcuna risposta dal superserver in quanto non ha modo di rilevarla avendo disabilitato l'ascolto di nuove richieste di tale servizio.



Nel momento però in cui il primo client chiude la connessione, il server a lui connesso viene terminato e il superserver, riabilitando l'ascolto di nuove richieste, alloca il tcpServer per il client che non aveva ancora ottenuto risposta.



TcpServer behavior in no-wait mode

Durante una comunicazione TCP no-wait, per ciascun client che effettua una richiesta viene allocato un TCP Server che risponde ai loro messaggi, tutto ciò avviene parallelamente.