UNIVERSITÀ DI PISA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA
SPECIFICHE PROGETTO A.A. 2021/2022

Reti informatiche, cod. 545II, 9 CFU

Prof. Giuseppe Anastasi, Ing. Francesco Pistolesi

Si richiede di progettare un'applicazione distribuita basata su paradigma ibrido peer-to-peer e client-server che implementi una moderna applicazione di instant messaging con notifiche, chat di gruppo, file sharing e messaggistica offline.

1. DESCRIZIONE GENERALE

L'applicazione di instant messaging da implementare è composta da un **server** e da un insieme di **device**. L'applicazione consente agli utenti (uno per device) di comunicare con altri utenti e condividere file. La comunicazione avviene tramite scambio di messaggi. Un tipico scenario applicativo è illustrato in Fig. 1.

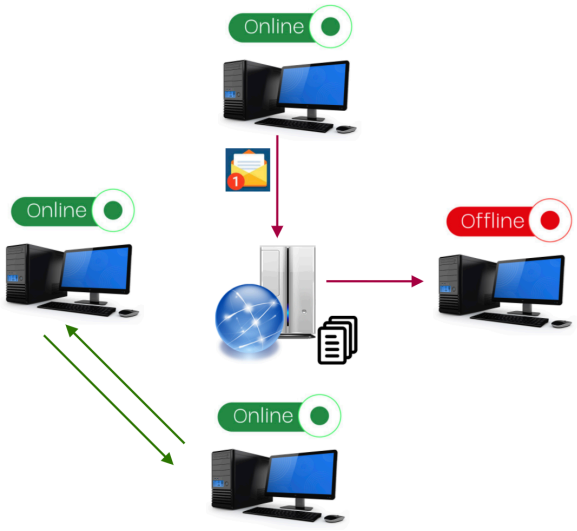

Figura 1: Scenario composto da quattro device, di cui tre online e uno offline.

Per poter usare l'applicazione la prima volta, un utente deve effettuare un'operazione di *signup*, contattando il server per creare un account costituito da username e password. Una volta in possesso di un account, l'utente può effettuare il *login*, accedendo così al menu dell'applicazione. Da quel momento, l'utente è online fintantoché non esegue il *logout*.

Un utente ha una rubrica contenente una lista di username di altri utenti. La rubrica è memorizzata su file. L'utente può avviare una conversazione scegliendo un *utente destinatario* dalla rubrica, al quale può poi inviare un messaggio. Se è la prima volta che l'utente invia un messaggio all'utente destinatario, l'applicazione dell'utente invierà il messaggio al server, il quale stabilirà se l'utente destinatario è online tramite un registro costituito da entry del tipo mostrato in Fig. 2, dove user_dest è uno username, port è la porta alla quale l'utente è raggiungibile, timestamp_login e timestamp_logout sono il timestamp di login e logout, rispettivamente.

user_dest	port	timestamp_login	timestamp_logout

Figura 2: Struttura di entry del registro del server.

Se nel registro è presente una entry con lo username del destinatario (user_dest), priva di timestamp di logout, l'utente destinatario è online. Il server recapiterà quindi il messaggio all'utente destinatario. Se l'invio va a buon fine, il server invia all'utente che ha iniziato la conversazione un messaggio contenente un ack di avvenuta ricezione del messaggio da parte dell'utente destinatario, comprensivo della porta (port) sulla quale l'utente destinatario è connesso. Se il server non riesce a inviare il messaggio, riprova per 3 volte, dopodiché memorizza il messaggio, aggiorna la entry relativa a user_dest ponendo il campo timestamp_logout al timestamp corrente, e invia all'utente che aveva inviato il messaggio un ack di avvenuta memorizzazione del messaggio, ma mancato recapito. Se l'utente continua a inviare messaggi a user_dest, questi saranno memorizzati sul server finché user_dest non effettuerà di nuovo il login. A quel punto, su richiesta di user_dest, il server recapiterà a user_dest i messaggi bufferizzati, inviando una conferma di avvenuta ricezione ai corrispondenti utenti mittenti se l'invio va a buon fine[1].

Implementare un protocollo che:

a) permetta al server di sapere quali utenti sono online e quali non lo sono;
b) di rilevare i login e i logout per far sì che i messaggi siano recapitati o bufferizzati nel caso il destinatario sia online od offline, rispettivamente.

Il protocollo può far uso di scambio di messaggi, registri, comunicazioni col server, oppure coordinazione fra device. Descrivere il protocollo implementato nella relazione in termini di formato dei pacchetti, interazioni (tramite un diagramma), e breve descrizione che ne illustri in maniera critica pregi e difetti.

Un utente che ha iniziato una conversazione con un utente destinatario, può includere nella conversazione altri utenti destinatari, solo se questi sono online, iniziando una chat di gruppo. I messaggi inviati dai vari utenti appartenenti alla chat di gruppo sono inviati automaticamente a tutti gli utenti della chat di gruppo.

Durante una chat, un utente può condividere anche file, inviandoli all'utente destinatario. Il protocollo di invio dei file è a discrezione dello studente. Nel caso di chat di gruppo, lo studente deve implementare un protocollo di condivisione del file che non coinvolga il server. Descrivere il protocollo nella documentazione.

L'applicazione distribuita da sviluppare deve implementare quanto descritto nel Paragrafo 1. Le scelte progettuali devono essere spiegate in una **relazione di una/due pagine**, font Arial, dimensione 12 pt, prediligendo l'uso di figure e schemi intuitivi (anche tracciati a mano e fotografati), limitando il testo. Nella relazione devono essere evidenziati pregi e difetti delle scelte progettuali effettuate, in termini di efficienza, aspetti critici, scalabilità, e così via.

[1] Quanto user_dest è offline, il server può ricevere messaggi destinati a user_dest provenienti da più utenti. Tutti questi messaggi vengono bufferizzati dal server e saranno inviati a user_dest quando questi lo richiederà.

1.1 DEVICE

Un **device** è mandato in esecuzione come segue:

`./dev <porta>`

dove **`<porta>`** è la porta associata al device.

Dopo essere stato mandato in esecuzione, il processo **dev** richiede username e password, separati da uno spazio. A login avvenuto, il processo mostra a video una breve guida dei comandi. Quando riceve un comando da standard input, oppure dopo la ricezione di un messaggio dalla rete da parte di altri device (o dal server), il device deve mostrare a video cosa sta facendo[2].

All'avvio, i comandi accettati dal device sono solo **`signup`** e **`in`**, i cui compiti sono i seguenti:

`signup` ***`username password`***
Permette a un utente di creare un account sul server, caratterizzato da username e password.

`in` ***`srv_port username password`***
Permette a un device di richiedere al server la connessione al servizio. Il server è in ascolto su localhost, sulla porta *srv_port*. Quando riceve il comando **`in`** da un device, il server registra l'avvenuta connessione dell'utente e il timestamp corrente. Le informazioni possono essere tenute in memoria o salvate su file.

Dopo l'esecuzione del login, l'applicazione mostra il *menu iniziale*, dove sono disponibili i seguenti comandi:

`hanging`
Permette all'utente di ricevere la lista degli utenti che gli hanno inviato messaggi mentre era offline. Tali messaggi sono detti *messaggi pendenti*. Per ogni utente, il comando mostra username, il numero di messaggi pendenti in ingresso, e il timestamp del più recente.

Il formato dei messaggi e il protocollo di comunicazione sono scelte progettuali che devono essere giustificate nella relazione.

`show` ***`username`***
Consente a un utente *u* di ricevere dal server i messaggi pendenti a lui inviati, mentre era offline, dall'utente *username*. Una volta inviati tali messaggi all'utente *u*, il server invia una notifica di avvenuta lettura all'utente *username* se *username* è online, altrimenti mantiene l'informazione e invia la notifica non appena l'utente *username* aprirà una chat con *u*.

`chat` ***`username`***
Avvia una chat con l'utente username. La chat contiene la cronologia dei messaggi. Ogni messaggio è preceduto da '*' se non ancora recapitato a *username*, oppure '**' se recapitato. Per inviare un messaggio all'utente *username*, è sufficiente digitarlo e premere invio. Per uscire, occorre digitare '\q+INVIO' e si torna al menu iniziale. Per aggiungere un partecipante, e creare quindi una chat di gruppo, occorre digitare '\u+INVIO'. L'applicazione mostrerà una lista con gli *username* degli utenti online. Per aggiungere un partecipante, digitare '\a username+INVIO'. Da quel momento, ogni messaggio viene spedito a tutti gli utenti inclusi nella chat di gruppo. Il protocollo utilizzato può essere sia TCP che UDP. Specificare nella relazione le motivazioni della scelta.

[2] Il device lavora cioè in modalità *verbose*, e informa l'utente continuativamente su ciò che sta facendo.

share ***file_name***
Invia il file *file_name* (presente nella current folder) al device su cui è connesso l'utente o gli utenti con cui si sta chattando. Il protocollo di condivisione del file non deve coinvolgere il server.

out
Il device richiede una disconnessione dal network. Quando un utente digita il comando out, viene inviato un messaggio al server il quale memorizza l'istante di uscita dell'utente nella relativa entry. Da quel momento, i messaggi inviati all'utente sono memorizzati sul server, assieme al timestamp di invio e di ricezione.

1.2 SERVER

Il server si avvia col seguente comando:

```
./serv [port]
```

dove **[porta]** è la porta associata. Il parametro è opzionale: se l'utente non lo inserisce, il server si avvia sulla porta 4242.

All'avvio, i comandi disponibili sono:

- **help**
- **list**
- **esc**

Un esempio di esecuzione è il seguente:

```
$ ./serv
```

il cui output potrebbe essere simile a quello mostrato in Fig. 3.

```
****************************** SERVER STARTED ********************************
Digita un comando:

1) help --> mostra i dettagli dei comandi
2) list --> mostra un elenco degli utenti connessi
3) esc --> chiude il
```

Figura 3: Esempio di esecuzione del server.

I comandi accettati da tastiera dal server sono i seguenti:

help
Mostra una breve descrizione dei comandi.

list
Mostra l'elenco degli utenti connessi alla rete, indicando username, timestamp di connessione e numero di porta nel formato "username*timestamp*porta".

esc
Termina il server. La terminazione del server non impedisce alle chat in corso di proseguire. Se il server è disconnesso, nessun utente può più fare login. Gli utenti online, qualora decidano di uscire, salvano l'istante di disconnessione. Quando il server si riconnette, gli utenti che si riconnettono dopo essersi disconnessi a server disattivo inviano l'istante di disconnessione salvato.

Supporre che i device possano disconnettersi improvvisamente senza effettuare `out`. Spiegare brevemente nella documentazione la politica utilizzata per gestire le disconnessioni improvvise e le conseguenze che comportano.

REQUISITI

- I dati sono scambiati tramite **socket**. Se non si definisce un formato di messaggio, prima di ogni scambio, il ricevente deve essere informato su **quanti byte** leggere dal socket.
- È possibile usare **strutture dati a piacere**. Gli aspetti non dettagliatamente specificati in questo documento possono essere implementati liberamente, spiegando ciò nella documentazione.
- Usare gli **autotools** (comando **make**) per la compilazione del progetto. Usare la breve guida disponibile sul sito Elearn del corso.
- Il codice **deve essere indentato** e **commentato in ogni parte**: significato delle variabili, processazioni, ecc. I commenti possono essere evitati nelle parti banali del codice.
- Va prodotta una **breve relazione di 1-2 pagine** (font Arial, size 12 pt) in cui spiegare, soprattutto tramite figure o diagrammi brevemente commentati, le scelte fatte circa le parti lasciate a scelta dello studente, le strutture dati usate e la struttura dei messaggi di rete (campi, numero di byte, ecc.).
- Scrivere uno **script che compili il progetto, mandi in esecuzione il server e avvii 3 device**, bloccati sul login, su finestre diverse del terminale.

CONSEGNA

Il progetto deve essere caricato in **formato .zip** sul sistema elearn.ing.unipi.it (sulla pagina del corso) **non oltre le 72 ore che precedono il giorno dell'esame**, usando le credenziali di Ateneo per l'accesso. Per ogni appello, sarà creata una nuova sezione per le consegne. Prima di caricare il progetto, **testare il codice su una macchina Debian 8**.

VALUTAZIONE

Il progetto è visionato testato e valutato prima del giorno dell'esame. Durante l'esame, può essere richiesta l'esecuzione del programma per spiegare alcune parti. Possono essere fatte domande sia su specifiche parti del codice che sulle scelte fatte.

La valutazione del progetto prevede le seguenti fasi:

1. **Compilazione del codice**
   Il codice va compilato con l'opzione **`-Wall`** che mostra i vari warning. Non vi dovranno essere warning o errori. L'opzione **`-Wall`** va abilitata anche durante lo sviluppo del progetto, interpretando i messaggi forniti dal compilatore. Se il progetto non compila o genera molteplici errori a runtime non è valutabile;
2. **Esecuzione dell'applicazione**
   In questa fase si verifica il funzionamento dell'applicazione e il rispetto delle specifiche;
3. **Analisi del codice sorgente**
   Può essere richiesto di spiegare parti del codice e apportarvi semplici modifiche durante l'esame.

FUNZIONI DI UTILITÀ

Una documentazione di alto livello per le funzioni necessarie per leggere e scrivere file a blocchi possono essere visualizzate al seguente indirizzo:

https://www.programiz.com/c-programming/c-file-input-output