NOME: Luca COGNOME: Sannino
MATRICOLA: 1542194

PROGETTO DI SISTEMI OPERATIVI (N.O.)

---------- SERVIZIO DI MESSAGGISTICA ----------

--- SPECIFICHE DEL PROGETTO:

Il servizio di messaggistica è realizzato attraverso l'ausilio di due componenti: Client e Server.
L'applicazione Client permette ad un utente, che abbia creato un account protetto da una password, o che abbia effettuato il login nel caso si sia già registrato precedentemente, di inviare messaggi verso gli altri utenti già iscritti al servizio e di leggere, ed eventualmente eliminare, i messaggi ricevuti.
L'applicazione Server si occupa quindi sia di conservare e di aggiornare la lista degli utenti registrati al servizio, in modo da riconoscere gli utenti che effettuano il login con una password corretta, sia di ricevere i messaggi scritti dagli utenti e di archiviarli nella casella personale dei loro rispettivi destinatari, in modo che possano essere poi inviati ai loro proprietari per essere letti su richiesta o per richiederne la rimozione.

--- SCELTE DI PROGETTO:

COMUNICAZIONE CLIENT-SERVER:

Client e Server comunicano tra loro grazie all'utilizzo delle socket che, servendosi del protocollo TCP , garantiscono una connessione sicura ed affidabile. Il Server processa le richieste dei Client sequenzialmente, ma nel frattempo può tenere una coda fino ad un massimo di cento richieste pendenti. Dato che il Client non ha bisogno di accettare chiamate, solo il socket del Server effettua il binding.
Le socket comunicano localmente (l'indirizzo IP del Server è quello corrispondente a LOCALHOST), attraverso la porta n.1999.
Per scambiarsi dati tra loro, Server e Client sfruttano un costrutto "pacchetto", al cui interno vengono salvate informazioni vitali, come il nome e la password dell'utente che desidera registrarsi o loggarsi (costrutto "utente"), o come i messaggi stessi (costrutto "messaggio"), che vengono suddivisi in quattro parti: oggetto, testo, mittente e destinatario. "pacchetto" contiene anche un char "istruzione", che viene utilizzato dal Client per comunicare le intenzioni dell'utente al Server e da quest'ultimo per segnalare al primo l'esito delle sue operazioni, un array di costrutti "messaggio", utilizzato dal Server per inviare tutti i messaggi al proprietario che li richiede, e infine un int, utilizzato dal Client per segnalare al Server il messaggio che desidera eliminare.
Nel caso uno dei due componenti abbia problemi nel ricevere o nell'inviare un pacchetto, esso allora si disconnette immediatamente dall'altro.

SEGNALI:

Sia Client che Server sono in grado di catturare segnali inviati dal sistema.
In entrambe le applicazioni viene infatti inizializzata una maschera a cui vengono aggiunti i diversi

tipi di segnali che devono essere bloccati. Successivamente ad ogni tipologia di segnale viene assegnata una nuova azione di risposta che va a sostituire quella di default.
I segnali SIGHUP, SIGINT, SISQUIT, SIGILL, SIGSEGV e SIGTERM fanno terminare l'applicazione, non prima che quest'ultima riconosca e stampi su schermo il segnale che l'ha uccisa e che si sia disconnessa.
SIGALARM fa invece disconnettere un'applicazione nel caso attenda troppo la risposta dell'altra, evitando che si crei una situazione di stallo eventualmente infinita.
SIGPIPE invece viene espressamente ignorato in quanto se una delle due applicazioni si disconnette durante uno scambio di dati, l'altra la seguirà a ruota quando scatterà il suo SIGALARM.
La maschera ovviamente non può proteggere le applicazioni da SIGKILL, che causerà la terminazione immediata del processo che lo riceve.

INTEGRITA' DEI DATI:
Per evitare che un segnale di terminazione inaspettato giunga mentre Server sta creando nuovi file o scrivendo informazioni vitali, è importante fare in modo che la maschera blocchi momentaneamente i segnali in arrivo in modo da non compromettere queste operazioni delicate.
A fare da cornice a queste sezioni di codice critiche sono state quindi inserite due funzioni, di cui la prima ha il compito di bloccare le tipologie di segnali che erano state aggiunte alla maschera, la seconda di sbloccarle.

--- FUNZIONAMENTO DEL PROGETTO:

REGISTRAZIONE:
Dopo aver fatto in modo che l'utente inserisca in input un nome ed una password che soddisfano il numero necessario di caratteri, e dopo essersi collegato al Server, Client invia a quest'ultimo un pacchetto con la richiesta di registrare un nuovo utente dallo stesso nome e dalla stessa password di quelli dati in input.
Una volta ricevuto il pacchetto, Server controlla nella directory "./Utenti" se esiste già un file con lo stesso nome dell'utente: in caso positivo, il Server fa sapere al Client che il nome è già stato usato, comunicando il fallimento della registrazione. In caso negativo, invece, Server crea un nuovo file con il nome dell'utente, scrivendoci dentro la password.
Dopodichè crea nella directory "./Messaggi" una nuova cartella con il nome dell'utente, dove all'interno saranno conservati i suoi messaggi, e comunica a quest'ultimo la buona riuscita della registrazione.
Nel caso la scrittura della password o la creazione della cartella dovessero fallire, Server elimina il file appena creato e riporta il fallimento della registrazione.
Terminata l'operazione, il Client si disconnette dal Server e viceversa.

LOGIN:
Dopo aver fatto in modo che l'utente inserisca in input un nome ed una password che soddisfano il numero necessario di caratteri, e dopo essersi collegato al Server, Client invia a quest'ultimo un pacchetto con la richiesta di login per un utente che si era già registrato con lo stesso nome e la stessa password di quelli dati in input.

Una volta ricevuto il pacchetto, Server controlla nella directory "./Utenti" se esiste già un file con lo stesso nome dell'utente: in caso negativo, il Server fa sapere al Client che l'utente non risulta registrato, comunicando il fallimento del processo di login. In caso positivo, invece, Server apre il file per leggere la password e confrontarla con quella all'interno del pacchetto. Il successo del processo di login dipende quindi dall'esito del confronto.
Terminata l'operazione, il Client si disconnette dal Server e viceversa.
Se il login è andato a buon fine, l'utente ha ora la possibilità di scrivere nuovi messaggi, leggere quelli ricevuti e, eventualmente, eliminarli.

SCRITTURA MESSAGGI:
Dopo aver fatto in modo che l'utente inserisca in input il destinatario, l'oggetto e il testo del suo messaggio, soddisfacendo il numero necessario di caratteri per ciascuno di essi, e dopo essersi collegato al Server, Client invia a quest'ultimo un pacchetto contenente il messaggio appena digitato dall'utente.
Una volta ricevuto il pacchetto, Server controlla nella directory "./Messaggi" se esiste già una cartella con lo stesso nome del destinatario: in caso negativo, il Server fa sapere al Client che il destinatario non risulta registrato, comunicando che non è stato possibile inviare il messaggio. In caso positivo, invece, Server conta i file (ognuno contenente un messaggio) all'interno della cartella. Se la cartella è vuota, Server crea un nuovo file, lo nomina "1" e scrive al suo interno il messaggio. Se invece ci sono già altri file, Server crea un nuovo file dandogli come nome il numero trovato precedentemente dalla conta ma sommato di 1, poi ci scrive dentro il messaggio. Infine, se il numero di messaggi già presenti è 100, Server comunica al Client che la casella del destinatario è piena e che non è stato possibile inviare il messaggio.
I messaggi sono numerati da 1 a 100 per garantire l'ordine cronologico di arrivo quando vengono inviati al loro proprietario per essere letti.
Quando un messaggio viene scritto, i suoi contenuti vengono divisi tra mittente, oggetto e testo.
Se il messaggio viene scritto correttamente nella cartella del destinatario, il Server comunica al Client la buona riuscita dell'operazione e ognuno di disconnette dall'altro.

LETTURA MESSAGGI:
Dopo essersi collegato al Server, Client invia a quest'ultimo un pacchetto contenente la richiesta dell'utente di leggere i suoi messaggi.
Una volta ricevuto il pacchetto, Server controlla la cartella dell'utente all'interno di "./Messaggi". Dopo aver contato il numero di messaggi per inizializzare un contatore, Server fa il ciclo di tutti i file all'interno della cartella, copiando il loro contenuto all'interno di un array di struct "messaggio".
A fine operazione, l'array verrà spedito al Client che procederà a stampare su schermo la lista di tutti i messaggi ricevuti.
A fine operazione, Server e Client si disconnettono l'uno dall'altro.

RIMOZIONE MESSAGGI:
Dopo aver fatto in modo che l'utente inserisse un numero compreso tra 1 e il numero di messaggi che ha ricevuto, e dopo essersi collegato al Server, Client invia a quest'ultimo un pacchetto

contenente la richiesta dell'utente di eliminare il messaggio dallo stesso numero di quello dato in input.
Una volta ricevuto il pacchetto, Server controlla la cartella dell'utente all'interno di "./Messaggi". Dopo aver eliminato il messaggio richiesto dall'utente, Server procede a rinominare i messaggi seguenti a quello eliminato, dandogli come nuovo nome il loro vecchio numero sottratto di uno. In questo modo, quando verrà scritto un nuovo messaggio che avrà come destinatario l'utente corrente, quel messaggio non verrà nominato con lo stesso numero di quello appena eliminato, ma sarà salvato in fondo alla lista, in modo da mantenere l'ordine cronologico di arrivo.
Se l'operazione di eliminazione avviene correttamente, il Server fa sapere al Client la buona riuscita dell'operazione, per poi disconnettersi l'uno dall'altro.

--- ISTRUZIONI PER L'USO:

CLIENT:

Il Client è composto dai seguenti files sorgente:

- ClientUtil.h
- ClientUtil.c
- Client.c

Per compilarlo correttamente, basta digitare la seguente formula sul prompt dei comandi:
gcc –o Client Client.c -lm –std=gnu99
Per avviarlo, invece:
./Client
(NOTA: Il programma è stato progettato per funzionare esclusivamente su macchine Linux).

SERVER:

Il Server è composto dai seguenti files sorgente:

- ServerUtil.h
- ServelUtil.c
- Server.c

In addizione, sono presenti anche due cartelli vitali per il funzionamento del programma:

- Utenti
- Messaggi

Per compilarlo correttamente, basta digitare la seguente formula sul prompt dei comandi:
gcc –o Server Server.c –lm –std=gnu99
Per avviarlo, invece:
./Server
(NOTA: Il programma è stato progettato per funzionare esclusivamente su macchine Linux).