Laboratorio di TPSI - Chat di Gruppo Socket TCP

Istituto di Istruzione Superiore "Archimede" - Treviglio

Autori: Biestro Edoardo, Medri Riccardo

Abstract

Il progetto consiste nella realizzazione di un'applicazione di chat di gruppo basata sull'architettura client-server. Il sistema permette a più utenti di connettersi contemporaneamente e scambiare messaggi in tempo reale. Il software è stato sviluppato in linguaggio Python utilizzando i socket TCP per la comunicazione di rete, il formato JSON per la strutturazione dei dati e il multi-threading per la gestione della concorrenza. Nonostante il software sia funzionale, l'obiettivo di connettere tutti i PC della classe è stato limitato dalla configurazione della rete scolastica, che suddivide i terminali su sottoreti diverse.

Introduzione

L'obiettivo del laboratorio era creare una chat di gruppo che permettesse a tutti i PC dell'aula di connettersi a un server centrale e comunicare tra loro.

La scelta degli strumenti è ricaduta su **Python** (versione 3.8 o superiore) per la sua versatilità e per la presenza di una libreria standard completa per la gestione delle reti (socket) e dei thread (threading). Per lo scambio dei dati è stato scelto il formato **JSON**, che permette di incapsulare in un'unica stringa leggibile sia il contenuto del messaggio che i metadati (mittente, IP).

Materiale e metodi

Strumenti di sviluppo

Per lo sviluppo del codice è stato utilizzato un IDE per Python (es. Visual Studio Code o IntelliJ IDEA con plugin Python). Le librerie utilizzate sono esclusivamente quelle standard di Python, garantendo la portabilità senza dipendenze esterne:

- `sys, os`: Per la gestione dei percorsi e del sistema.
- `socket`: Per la creazione delle connessioni TCP/IP.
- `threading`: Per l'esecuzione parallela di processi (invio/ricezione).
- `json`: Per la codifica e decodifica dei messaggi.

Descrizione del codice

Il progetto è strutturato in tre moduli principali:

1. **Utils (`utils.py`):** Contiene le costanti di configurazione (Porta 5000, Buffer 1024, Encoding UTF-8) e le funzioni comuni.
    - `create_json_msg`: Crea un oggetto JSON contenente il dizionario con chiavi "from" (nome e IP) e "message".
    - `decode_json_msg`: Converte i byte ricevuti in dizionario Python.
2. **Server (`server_main.py` e `chat_server.py`):** Il server ascolta su tutte le interfacce di rete (`0.0.0.0`). Utilizza la funzione `broadcast` per inoltrare i messaggi a tutti i client eccetto il mittente.
    - Viene utilizzato un oggetto `lock = thr.Lock()` per gestire in sicurezza l'accesso concorrente al dizionario `clients` che memorizza le connessioni attive.
    - Gestisce l'ingresso e l'uscita degli utenti inviando messaggi di notifica automatica al gruppo.
3. **Client (`chat_client.py`):** Il client richiede all'avvio username e IP del server (default localhost).
    - Viene avviato un thread separato (`threading.Thread`) con target la funzione `receive`, che ascolta costantemente i messaggi in arrivo senza bloccare l'input dell'utente.
    - Il ciclo principale gestisce l'input da tastiera, codifica il messaggio in JSON e lo invia al server.

Risultati e discussione

Il codice prodotto è risultato funzionante e privo di errori di sintassi. Il protocollo di comunicazione TCP garantisce l'integrità dei dati e la struttura JSON permette una chiara identificazione del mittente.

Tuttavia, durante la fase di test in laboratorio, è emersa una problematica infrastrutturale. L'obiettivo iniziale prevedeva la connessione di tutti i PC della classe, ma ciò non è stato completamente possibile. La maggior parte dei computer del laboratorio si trova su reti o VLAN diverse e, a causa delle restrizioni di rete o della configurazione delle sottoreti, non è stato possibile stabilire una connessione diretta tra i client e il server ospitato su uno dei PC.

Il sistema funziona perfettamente in ambiente `localhost` o all'interno della stessa sottorete LAN, dove il broadcast e la gestione dei thread operano come previsto.

Conclusione

L'obiettivo didattico di sviluppare un'applicazione multithread client-server è stato raggiunto con successo dal punto di vista software. Il sistema gestisce correttamente connessioni multiple, disconnessioni impreviste e formattazione dei dati.

Il limite riscontrato nella comunicazione tra PC diversi non è imputabile al codice, ma alla topologia della rete scolastica che impedisce il routing tra i dispositivi coinvolti. Per superare questo ostacolo in futuro, sarebbe necessario ospitare la parte server su una macchina accessibile da tutte le sottoreti (es. un server cloud o una macchina in DMZ).

Bibliografia

- Documentazione ufficiale Python (socket, threading, json).
- Materiale didattico fornito dal docente.
- Risorse online consultate per la gestione dei socket TCP.