Progetto di Reti di Calcolatori

Prof. Umberto Scafuri

A.A. 2024/2025

PROPONENTI:
Salvatore D'Onofrio 0124/002723
Saverio Pio Pipola 0124/002726
Mattia Jué 0124/002806

INDICE

1. Descrizione del Progetto

Obiettivo del Progetto

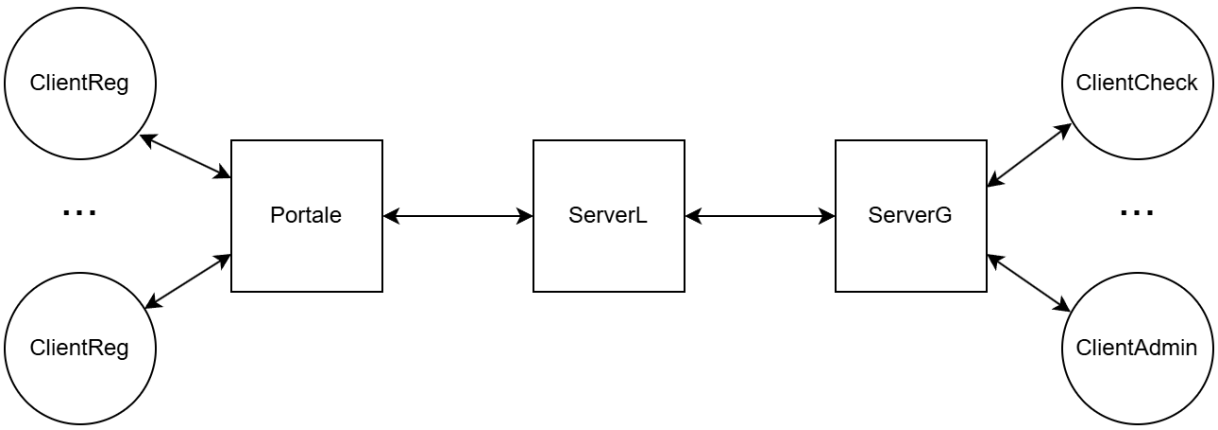
Il progetto consiste nello sviluppo di un sistema di gestione delle licenze software basato su un'architettura client/server. L'obiettivo è garantire che i software registrati possano essere utilizzati solo se dotati di una licenza valida, evitando usi non autorizzati.

Il sistema permette di registrare, verificare e invalidare licenze attraverso una comunicazione tra un client e un server. Il server gestisce le richieste e mantiene le informazioni delle licenze in un file JSON, mentre il client invia comandi per effettuare operazioni sulle licenze.

Caratteristiche Principali

- **Registrazione delle licenze:** una nuova licenza viene memorizzata con data di attivazione e durata in mesi.
- **Verifica della validità:** il sistema calcola la data di scadenza sommando i mesi di validità alla data di attivazione.
- **Invalidazione delle licenze:** è possibile disattivare una licenza, rendendola inutilizzabile.
- **Formato JSON** per il salvataggio delle licenze, garantendo semplicità e leggibilità dei dati.

2. Architettura del Progetto



L'architettura del progetto si fonda su un sistema distribuito, in cui vari componenti comunicano tra loro utilizzando il protocollo TCP. Questo sistema è stato progettato per la gestione centralizzata delle licenze software, e comprende diversi client concorrenti (ClientReg, ClientCheck e ClientAdmin) che interagiscono con tre server distinti: Portale, ServerL e ServerG. La struttura distribuita consente di gestire in modo efficiente le licenze, assicurando scalabilità, robustezza e affidabilità nelle operazioni di registrazione, verifica e invalidazione.

2.1 Schemi dell'Architettura

Il sistema progettato include i seguenti componenti principali:

- **Client:** Ci sono diverse tipologie di client che interagiscono con il sistema, ognuno con funzioni specifiche. I client sono suddivisi in:
  ClientReg: si occupa della registrazione delle licenze.
  ClientCheck: verifica lo stato di una licenza.
  ClientAdmin: ha il compito di invalidare una licenza.
- **Server Portale:** Funziona come un intermediario tra il ClientReg e il server centrale. Riceve le richieste dai client e le instrada al server centrale per l'elaborazione, facilitando la comunicazione tra i diversi client e il server di gestione delle licenze.
- **Server Centrale (ServerL):** È il cuore del sistema, responsabile della gestione delle licenze. Le licenze sono memorizzate in un file JSON, e il server centrale si occupa di elaborare le richieste di registrazione, verifica e invalidazione delle licenze.
- **ServerG:** Rappresenta un ulteriore livello di gestione. Si occupa della comunicazione tra il server centrale e i client di verifica (ClientCheck) e di amministrazione (ClientAdmin), coordinando l'interazione tra questi componenti.

2.2 Schemi del Protocollo Applicazione

Il progetto è stato sviluppato utilizzando il protocollo TCP (Transmission Control Protocol), un protocollo di trasporto ampiamente adottato nelle reti di calcolatori per la sua affidabilità e performance.
Uno dei principali vantaggi di TCP è la sua affidabilità. Grazie all'uso di numeri di sequenza e riconoscimento, TCP garantisce che tutti i dati vengano ricevuti nell'ordine corretto e senza alcuna perdita. Inoltre, essendo un protocollo connection-oriented, prima della trasmissione dei dati viene stabilita una connessione logica tra i due endpoint. Questo assicura un trasferimento più sicuro ed efficiente dei dati.
Un altro vantaggio significativo è la presenza di meccanismi di controllo della congestione. Questi meccanismi impediscono il sovraccarico della rete, garantendo una gestione equa delle risorse e prevenendo rallentamenti o interruzioni nella comunicazione.

3. Tecnologie e Soluzioni Utilizzate

3.1 Linguaggio di Programmazione

Il progetto è stato sviluppato in Python grazie alla sua sintassi chiara e alla vasta disponibilità di librerie. Python è stato scelto anche per la sua capacità di gestire facilmente la programmazione di rete tramite la libreria socket e per la sua facilità nell'implementazione delle funzionalità di gestione delle licenze.

3.2 Protocollo di Comunicazione

Il protocollo TCP è stato implementato in Python grazie alla libreria socket, che offre un insieme di funzioni per gestire le comunicazioni di rete in modo semplice ed efficace. Utilizzando il modulo socket, è stato possibile creare server e client che si connettono tramite porte TCP, scambiandosi messaggi in modo sicuro. Python semplifica notevolmente la gestione della connessione TCP, evitando la necessità di configurare manualmente i dettagli complessi del protocollo e garantendo una trasmissione affidabile dei dati.

3.3 Persistenza dei Dati

I dati relativi alle licenze (come identificatore, data di acquisto e durata di validità) sono memorizzati in un file JSON. Questo formato è stato scelto per la sua leggerezza e facilità di gestione, consentendo di accedere e aggiornare rapidamente lo stato delle licenze. Il file JSON funge da "database" persistente per il sistema, contenendo tutte le informazioni necessarie per la gestione delle licenze software.

3.4 Gestione delle Concorrenze

La gestione della concorrenza è un aspetto fondamentale del progetto, poiché consente di gestire più client simultaneamente senza compromettere le performance. Grazie all'uso della libreria threading di Python, ogni client che si connette al server viene gestito in un thread separato, permettendo l'esecuzione parallela delle operazioni senza blocchi o rallentamenti. Inoltre, per evitare conflitti nell'accesso al file JSON da parte di più thread, è stato implementato un meccanismo di locking, che garantisce che solo un thread alla volta possa modificare il file, assicurando così la consistenza dei dati.

4. Implementazione

4.1 Dettagli Implementativi dei Client

4.1.1 ClientReg (Client di Registrazione)

Il ClientReg è responsabile della registrazione delle licenze nel sistema. Quando un utente vuole registrare una nuova licenza software, il client raccoglie i dati necessari, come l'ID della licenza, la data di acquisto e la durata della validità, e invia queste informazioni al server.

- **Connessione:** Il client stabilisce una connessione con il **Server Portale** tramite il protocollo TCP, utilizzando un socket.
- **Invio dei dati:** Dopo aver ricevuto i parametri dall'utente, il client invia i dati al server sotto forma di messaggio JSON contenente l'ID della licenza, la data di acquisto e i mesi di validità.
- **Gestione della risposta:** Il client riceve la risposta dal server e la visualizza all'utente. Se la licenza viene registrata correttamente, il client mostrerà un messaggio di successo, altrimenti un messaggio di errore (ad esempio, se la licenza è già registrata).

4.1.2 ClientCheck (Client di Verifica)

Il ClientCheck è utilizzato per verificare la validità di una licenza. L'utente può inserire l'ID della licenza e il client invia una richiesta al server per ottenere lo stato della licenza (valida, scaduta, invalidata).

- **Connessione:** Il client si connette al ServerG tramite il protocollo TCP, utilizzando un socket.
- **Invio dei dati:** Il client invia al server un messaggio JSON contenente l'ID della licenza da verificare.
- **Gestione della risposta:** Il client riceve la risposta dal server e la visualizza all'utente.

4.1.3 ClientAdmin (Client di Amministrazione)

Il ClientAdmin permette all'amministratore di gestire le licenze in modo più avanzato, come invalidare licenze esistenti. Un amministratore può scegliere una licenza da invalidare, e il client invia la richiesta corrispondente al server.

- **Connessione:** Il client si connette al ServerG tramite il protocollo TCP, utilizzando un socket.
- **Invio dei dati:** L'amministratore seleziona una licenza da invalidare, e il client invia l'ID della licenza al server per l'operazione di invalidazione.
- **Gestione della risposta:** Dopo che il server ha processato la richiesta, il client riceve la risposta e la mostra all'utente. Se la licenza è stata invalidata con successo, il client fornirà una conferma, altrimenti segnalerà eventuali errori.

4.2 Dettagli Implementativi dei Server

I server vengono attivati sull'indirizzo localhost (127.0.0.1) e utilizzano le seguenti porte:

SERVERL_PORT: 5000
PORTALE_PORT: 6000
SERVERG_PORT: 7000

4.2.1 Portale

Il Server Portale funge da intermediario tra il ClientReg e il Server Centrale (ServerL) per la registrazione di una nuova licenza.

- **Connessione:** Il server ascolta le connessioni in ingresso dai client sulla porta specificata tramite il protocollo TCP, utilizzando un socket. Il client di registrazione si connette a questa porta per inviare i dati relativi alla licenza.
- **Gestione delle richieste:** Quando riceve una richiesta di registrazione (contenente i dati della licenza), il server verifica se i dati sono corretti e inoltra la richiesta al Server Centrale (ServerL) per l'elaborazione effettiva.
- **Risposta al client:** Dopo aver ricevuto una risposta dal Server Centrale, che include il risultato della registrazione (successo o errore), il server invia tale risposta al ClientReg. Nel caso di successo, il client visualizzerà un messaggio di conferma; in caso di errore,

verrà mostrato un messaggio che descrive il problema (ad esempio, licenza già registrata).

4.2.2 ServerG

Il ServerG gestisce le operazioni di verifica e invalidazione delle licenze e funge da intermediario tra il Server Centrale (ServerL) e i client di verifica e amministrazione (ClientCheck e ClientAdmin).

- **Connessione:** Il server ascolta le richieste dei client sulla porta specificata tramite il protocollo TCP, utilizzando un socket. ClientCheck e ClientAdmin si connettono a questa porta per inviare richieste di verifica e invalidazione.
- **Gestione delle richieste di verifica e invalidazione:** Quando riceve una richiesta di verifica (da ClientCheck) o di invalidazione (da ClientAdmin), il server inoltra la richiesta al Server Centrale (ServerL) per l'elaborazione. Il server può effettuare un controllo preliminare sui dati ricevuti prima di inoltrarli, ma la gestione delle licenze è delegata al ServerL.
- **Risposta ai client:** Dopo aver ricevuto una risposta dal Server Centrale, che contiene lo stato aggiornato della licenza (ad esempio, "valida", "scaduta" o "invalidata"), il server invia il risultato al client appropriato (ClientCheck o ClientAdmin). In caso di errore, ad esempio se una licenza non esiste, viene restituito un messaggio di errore.

4.2.3 ServerL (Server Centrale)

Il Server Centrale (ServerL) è il componente principale del sistema, responsabile della gestione delle licenze. Questo server memorizza le licenze in un file JSON e gestisce tutte le operazioni relative alle licenze, come la registrazione, la verifica e l'invalidazione.

- **Connessione:** Il server ascolta le richieste in ingresso sulla porta designata tramite il protocollo TCP, utilizzando un socket. Accetta connessioni sia dal Server Portale che dal ServerG per ricevere e rispondere alle richieste.
- **Gestione delle licenze:** Quando riceve una richiesta di registrazione, verifica o invalidazione della licenza, il server carica il file JSON contenente le licenze. Per la registrazione, il server aggiorna il file con i nuovi dati della licenza. In caso di verifica, confronta l'ID della licenza ricevuto con quelli già memorizzati nel file JSON e risponde con lo stato della licenza (valida, scaduta o invalidata). Per l'invalidazione, il server aggiorna lo stato della licenza nel file JSON, segnandola come "invalidata". Dopo aver eseguito l'operazione richiesta, il server risponde al client con il risultato dell'operazione (ad esempio, conferma di registrazione, stato della licenza o conferma di invalidazione).
- **Persistenza dei dati:** Ogni volta che una licenza viene registrata o invalidata, il server aggiorna il file JSON per mantenere la consistenza dei dati. L'aggiornamento è atomico, per evitare conflitti durante le operazioni concorrenti.

4.3 Dettagli Implementativi del File JSON

4.3.1 Struttura del File JSON

Il file JSON è organizzato come un oggetto contenente una serie di licenze, identificate da un ID univoco. Ogni licenza è rappresentata come un oggetto che contiene le seguenti informazioni:

- **ID della licenza:** identificatore unico della licenza.
- **Data di acquisto:** la data in cui la licenza è stata acquistata (formato "YYYY-MM-DD").
- **Validità:** il periodo di validità in mesi.
- **Stato della licenza:** un valore booleano che indica se la licenza è attiva (true) o invalidata (false).

4.4 Frammenti di Codice per i Client

4.4.1 clientReg.py

Connessione al Server

Il client si connette al server tramite il protocollo TCP sulla porta 6000 (definita nella variabile SERVER_PORT). La connessione è realizzata utilizzando il modulo socket di Python.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((SERVER_HOST, SERVER_PORT))  # Connessione al portale
```

Creazione del Messaggio JSON

Il client prepara un messaggio contenente le informazioni necessarie per registrare una licenza. Il messaggio è strutturato in formato JSON per facilitare la comunicazione tra il client e il server. Il messaggio include:

- **comando:** Indica il tipo di operazione (in questo caso, "REGISTRA").
- **licenza:** Il codice identificativo della licenza.
- **data_acquisto:** La data di acquisto della licenza nel formato YYYY-MM-DD.
- **validita_mesi:** La durata della licenza espressa in mesi.

```
# Creiamo il messaggio in formato JSON con il codice della licenza e i dettagli
richiesta = {
    "comando": "REGISTRA", # Comando per registrare una nuova licenza
    "licenza": codice_licenza,
    "data_acquisto": data_acquisto,
    "validita_mesi": validita_mesi
}
```

Invio dei Dati al Server

Il messaggio viene convertito in formato JSON tramite `json.dumps()` e inviato al server utilizzando il metodo `sendall()` del socket.

```
# Convertiamo la richiesta in formato JSON e la inviamo al server
sock.sendall(json.dumps(richiesta).encode())
```

Gestione degli Errori

Il client è dotato di una gestione degli errori, che cattura eventuali eccezioni durante la connessione e la comunicazione con il server, restituendo un messaggio di errore utile per il debug.

```
except Exception as e:
    print(f"Errore durante la comunicazione con il server: {e}")
```

4.4.2 clientCheck.py

Molte operazioni, come la connessione al server, la creazione del messaggio JSON, l'invio dei dati al server e la gestione degli errori, sono sostanzialmente identiche a quelle già descritte nel codice precedente. Pertanto, queste parti non verranno ripetute, ma si darà per scontato che siano implementate in modo simile.

Ricezione della Risposta dal Server

Il client riceve la risposta dal server che contiene lo stato della licenza. La risposta può essere una delle seguenti:

```
"OK: Licenza valida"
"ERRORE: Licenza non trovata"
"ERRORE: Licenza scaduta"
"ERRORE: Licenza invalidata"
```

In base alla risposta ricevuta, il client visualizza un messaggio appropriato per l'utente.

```
# Ricezione della risposta dal serverG
risposta = sock.recv(1024).decode()

# Gestiamo la risposta per determinare la validità della licenza
if risposta == "OK: Licenza valida":
    print(f"Licenza {codice_licenza} valida.")
elif risposta == "ERRORE: Licenza non trovata":
    print(f"Licenza {codice_licenza} non trovata.")
elif risposta == "ERRORE: Licenza scaduta":
    print(f"Licenza {codice_licenza} scaduta.")
elif risposta == "ERRORE: Licenza invalidata":
    print(f"Licenza {codice_licenza} invalidata.")
else:
    print(f"Risposta imprevista dal server: {risposta}")
```

4.4.3 clientAdmin.py

Ricezione della Risposta dal Server

Dopo l'invio della richiesta, il client attende la risposta del server. A seconda della risposta ricevuta, viene mostrato un messaggio all'utente, che indica se l'operazione di invalidazione è stata completata con successo o se c'è stato un errore.

```
# Ricezione della risposta dal server
risposta = sock.recv(1024).decode()

if risposta == "OK: Licenza invalidata":
    print(f"Licenza {codice_licenza} invalidata con successo.")
else:
    print(f"Errore nell'invalidare la licenza {codice_licenza}: {risposta}")
```

4.5 Frammenti di Codice per i Server

4.5.1 portale.py

Gestione della Connessione Client-Server

Il server `Portale` è configurato per ascoltare le richieste sulla porta 6000. Quando un client si connette, viene gestito da un thread separato, il quale permette di gestire più connessioni contemporaneamente.

```
# Configurazioni
HOST = "localhost"
PORT = 6000

# Funzione per avviare il server
def avvia_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(5)
    print(f"Portale in ascolto su {HOST}:{PORT}")

    while True:
        conn, addr = server.accept()
        threading.Thread(target=gestisci_client, args=(conn, addr)).start()
```

Ricezione della Richiesta dal Client

Ogni client che si connette invia una richiesta che il server `Portale` riceve e decodifica. La richiesta è in formato JSON e contiene un comando (ad esempio, "REGISTRA") e i relativi dati, come il codice della licenza, la data di acquisto e la validità in mesi.

```
dati = conn.recv(1024).decode()  # Riceve la richiesta dal client
if not dati:
    break

# Decodifica della richiesta JSON
richiesta = json.loads(dati)
```

Elaborazione della Richiesta di Registrazione

Se il comando ricevuto è "REGISTRA", il server `Portale` estrae i dettagli della licenza dalla richiesta JSON e costruisce un nuovo messaggio JSON per inoltrarlo al server centrale (ServerL) che gestisce le operazioni di registrazione.

```
comando = richiesta.get("comando")
if comando == "REGISTRA":
    # Estrai i parametri dalla richiesta JSON
    codice_licenza = richiesta["licenza"]
    data_acquisto = richiesta["data_acquisto"]
    validita_mesi = richiesta["validita_mesi"]

    # Crea un messaggio per il serverL
    dati_serverL = json.dumps({
        "comando": "REGISTRA",
        "licenza": codice_licenza,
        "data_acquisto": data_acquisto,
        "validita_mesi": validita_mesi
    })
```

Inoltro della Richiesta al Server Centrale

Il server `Portale` invia la richiesta di registrazione al server centrale (ServerL) utilizzando la funzione `inoltra_richiesta()`, che si occupa di stabilire la connessione con il server centrale, inviare il messaggio e ricevere la risposta.

```
# Invia la richiesta al serverL
risposta = inoltra_richiesta(SERVER_L_HOST, SERVER_L_PORT, dati_serverL)

# Funzione per inoltrare le richieste ai server
def inoltra_richiesta(server_host, server_port, messaggio):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((server_host, server_port))
            s.sendall(messaggio.encode())
            risposta = s.recv(1024).decode()
        return risposta
    except Exception as e:
        return f"ERRORE: {str(e)}"
```

Invio della Risposta al Client

Dopo aver ricevuto la risposta dal server centrale, il server `Portale` invia la risposta al client, indicando se la registrazione è stata effettuata correttamente o se ci sono stati errori.

```
# Invia la risposta al client
conn.sendall(risposta.encode())
```

Avvio del Server

Il server `Portale` è progettato per avviarsi in modo continuo, restando in ascolto per nuove connessioni. Ogni nuova connessione viene gestita da un thread separato, permettendo al server di gestire più richieste contemporaneamente.

```
while True:
    conn, addr = server.accept()
    threading.Thread(target=gestisci_client, args=(conn, addr)).start()
```

4.5.2 serverG.py

Gestione della Connessione e Ricezione della Richiesta

Il server è configurato per ascoltare sulla porta 7000 e accetta le connessioni in entrata. Ogni client che si connette viene gestito da un thread separato, permettendo la gestione di più connessioni contemporaneamente.

```
# Configurazioni
SERVERG_PORT = 7000
SERVERL_HOST = "localhost"
SERVERL_PORT = 5000  # Porta del serverL per la gestione delle licenze
THREAD_POOL_SIZE = 10

# Funzione per avviare il server
def avvia_server():
    server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_sock.bind(('localhost', SERVERG_PORT))
    server_sock.listen(THREAD_POOL_SIZE)

    print(f"ServerG in ascolto sulla porta {SERVERG_PORT}")

    # Accetta le connessioni in parallelo utilizzando un pool di thread
    while True:
        client_sock, client_address = server_sock.accept()
        print(f"Connessione ricevuta da {client_address}")

        # Gestisce la connessione del client in un thread separato
        threading.Thread(target=handle_connection, args=(client_sock,)).start()
```

Quando il server riceve una connessione, viene creata una nuova connessione per ogni client, e il server legge i dati ricevuti.

```
# Riceve i dati dal client (in formato JSON)
data = client_sock.recv(1024).decode()
```

I dati ricevuti sono in formato JSON, quindi vengono decodificati e analizzati.

```
# Convertiamo i dati ricevuti in un dizionario
richiesta = json.loads(data)
```

Gestione della Richiesta del Client

Il server determina il tipo di operazione richiesta dal client (verifica della licenza o invalidazione). Se il comando è "VERIFICA" o "INVALIDA", il server procede, altrimenti invia una risposta di errore.

```
# Determiniamo il tipo di servizio richiesto (Verifica o Invalidazione)
if richiesta['comando'] == "VERIFICA":
    print("Servizio richiesto: verifica di validità della licenza")
elif richiesta['comando'] == "INVALIDA":
    print("Servizio richiesto: invalidazione della licenza")
else:
    client_sock.sendall("ERRORE: Comando non riconosciuto".encode())
    return
```

Inoltro della Richiesta al Server Centrale (ServerL)

Una volta che il server ha determinato il tipo di operazione richiesta, inoltra la richiesta al server centrale (ServerL) utilizzando la funzione `inoltra_richiesta_serverL()`. La comunicazione tra il serverG e ServerL avviene tramite una connessione TCP, dove il messaggio viene inviato e la risposta ricevuta.

```
# Inoltra la richiesta al serverL
risposta_serverL = inoltra_richiesta_serverL(data)

# Funzione per inoltrare la richiesta al serverL
def inoltra_richiesta_serverL(messaggio):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((SERVERL_HOST, SERVERL_PORT))
            s.sendall(messaggio.encode())
            risposta = s.recv(1024).decode()
        return risposta
    except Exception as e:
        return f"ERRORE: {str(e)}"
```

4.5.3 serverL.py

Gestione del File delle Licenze

Il server utilizza un file JSON (`licenze.json`) per memorizzare le licenze software. Se il file non esiste, viene creato inizialmente come file vuoto. Le funzioni `carica_licenze()` e `salva_licenze()` sono utilizzate per caricare e salvare il contenuto del file in modo sicuro, utilizzando un `Lock` per evitare condizioni di gara durante l'accesso concorrente al file.

```
# Carica il file JSON con le licenze
if not os.path.exists(LICENZE_FILE):
    with open(LICENZE_FILE, "w") as f:
        json.dump({}, f)

def carica_licenze():
    try:
```

```
        with LOCK, open(LICENZE_FILE, "r") as f:
            licenze = json.load(f)
            if not isinstance(licenze, dict):  # Controllo se è un dizionario
valido
                return {}
            return licenze
    except (json.JSONDecodeError, FileNotFoundError):
        return {}  # Se il file è vuoto, corrotto o inesistente, restituisce un
dizionario vuoto
```

Registrazione di una Nuova Licenza

La funzione `registra_licenza()` consente di registrare una nuova licenza nel sistema. Se la licenza esiste già, restituisce un errore. Altrimenti, aggiunge la licenza al file JSON con la data di acquisto e la durata in mesi.

```
def registra_licenza(software_id, data_acquisto, validita_mesi):
    licenze = carica_licenze()
    if software_id in licenze:
        return "ERRORE: Licenza già registrata"
    licenze[software_id] = {"data_acquisto": data_acquisto, "validita_mesi":
validita_mesi, "valida": True}
    salva_licenze(licenze)
    return "OK: Licenza registrata"

def salva_licenze(licenze):
    with LOCK, open(LICENZE_FILE, "w") as f:
        json.dump(licenze, f, indent=4)
```

Verifica della Validità di una Licenza

La funzione `verifica_licenza()` verifica se una licenza esiste nel sistema e se è ancora valida. Se la licenza non è valida o è scaduta, restituisce un errore. La data di scadenza viene calcolata aggiungendo i mesi di validità alla data di acquisto.

```
def verifica_licenza(software_id):
    licenze = carica_licenze()
    if software_id not in licenze:
        return "ERRORE: Licenza non trovata"

    licenza = licenze[software_id]

    if not licenza["valida"]:
        return "ERRORE: Licenza invalidata"

    # Converto la data di attivazione in formato datetime
    data_attivazione = datetime.strptime(licenza["data_acquisto"], "%Y-%m-%d")

    # Aggiungo i mesi di validità alla data di attivazione
    mesi_validita = licenza["validita_mesi"]
    data_scadenza = data_attivazione + timedelta(days=mesi_validita * 30)  #
Ogni mese ha 30 giorni

    # Ottengo la data odierna
    oggi = datetime.today()
```

```
    if oggi > data_scadenza:
        return "ERRORE: Licenza scaduta"

    return "OK: Licenza valida"
```

Invalidazione di una Licenza

La funzione `invalida_licenza()` consente di invalidare una licenza. Dopo aver trovato la licenza nel sistema, imposta il campo `"valida"` su `False` e salva il file aggiornato.

```
def invalida_licenza(software_id):
    licenze = carica_licenze()
    if software_id not in licenze:
        return "ERRORE: Licenza non trovata"

    licenze[software_id]["valida"] = False
    salva_licenze(licenze)
    return "OK: Licenza invalidata"
```

Gestione della Connessione con il Client

La funzione `gestisci_client()` gestisce la connessione con ogni client. Dopo aver ricevuto la richiesta dal client, il server decodifica il messaggio JSON, esegue l'operazione richiesta (registrazione, verifica o invalidazione della licenza) e invia la risposta al client.

```
def gestisci_client(conn, addr):
    print(f"Connessione da {addr}")
    while True:
        try:
            dati = conn.recv(1024).decode()
            if not dati:
                break

            richiesta = json.loads(dati)  # Parsing del JSON
            comando = richiesta.get("comando")

            print(f"Ricevuto comando: {comando}, dati: {richiesta}")  # DEBUG

            if comando == "REGISTRA":
                software_id = richiesta.get("licenza")
                data_attivazione = richiesta.get("data_acquisto")
                validita_mesi = richiesta.get("validita_mesi")

                if None in (software_id, data_attivazione, validita_mesi):
                    risposta = "ERRORE: Parametri mancanti per la registrazione"
                else:
                    risposta = registra_licenza(software_id, data_attivazione,
validita_mesi)

            elif comando == "VERIFICA":
                software_id = richiesta.get("licenza")

                if software_id is None:
                    risposta = "ERRORE: Parametri mancanti per la verifica"
                else:
                    risposta = verifica_licenza(software_id)

            elif comando == "INVALIDA":
                software_id = richiesta.get("licenza")
```

```
                if software_id is None:
                    risposta = "ERRORE: Parametri mancanti per invalidazione"
                else:
                    risposta = invalida_licenza(software_id)

            else:
                risposta = "ERRORE: Comando sconosciuto"

            conn.sendall(risposta.encode())
        except Exception as e:
            print(f"Errore: {e}")
            break
    conn.close()
```

4.5.4 licenze.json

Il file licenze.json è utilizzato dal serverL per caricare e gestire le informazioni relative alle licenze. Quando un client invia una richiesta per verificare, registrare o invalidare una licenza, il server consulta questo file per determinare lo stato della licenza e rispondere di conseguenza.

Il file viene letto e modificato tramite operazioni di lettura e scrittura in formato JSON, garantendo che le informazioni sulle licenze siano sempre aggiornate. Ogni volta che una licenza viene registrata, verificata o invalidata, il file viene aggiornato di conseguenza per riflettere lo stato corrente delle licenze.

```
{
    "LICENZA12345": {
        "data_acquisto": "2023-01-01",
        "validita_mesi": 12,
        "valida": true
    },
    "LICENZA67890": {
        "data_acquisto": "2022-06-01",
        "validita_mesi": 6,
        "valida": false
    },
    "LICENZA24680": {
        "data_acquisto": "2024-02-01",
        "validita_mesi": 24,
        "valida": true
    }
}
```

5. Manuale Utente

5.1 Istruzioni per la Compilazione

Prima di proseguire con le istruzioni per la compilazione, assicurarsi che Python sia installato sulla propria macchina.

Aprire il terminale, spostarsi nella directory del progetto ed eseguire i seguenti comandi:

`python serverL.py`

```
PS D:\Desktop\RETI\LicenzeSoftware> python serverL.py
ServerL in ascolto su localhost:5000
```

`python portale.py`

```
PS D:\Desktop\RETI\LicenzeSoftware> python portale.py
Portale in ascolto su localhost:6000
```

`python serverG.py`

```
PS D:\Desktop\RETI\LicenzeSoftware> python serverG.py
ServerG in ascolto sulla porta 7000
```

5.2 Istruzione per l'Esecuzione

5.2.1 ClientReg

Aprire il terminale, navigare fino alla directory del progetto ed eseguire il seguente comando, specificando come parametri il numero della licenza, la data di acquisto (nel formato "YYYY-MM-DD") e la durata della validità in mesi:

`python clientReg.py idLicenza "YYYY-MM-DD" nMesi`

```
PS D:\Desktop\RETI\LicenzeSoftware> python clientReg.py LICENZA99999 "2025-02-05" 6
Risposta dal server: OK: Licenza registrata
PS D:\Desktop\RETI\LicenzeSoftware>
```

5.2.2 ClientCheck

Aprire il terminale, navigare fino alla directory del progetto ed eseguire il seguente comando, specificando come parametri il numero della licenza:

`python clientCheck.py idLicenza`

```
PS D:\Desktop\RETI\LicenzeSoftware> python clientCheck.py LICENZA12345
Licenza LICENZA12345 scaduta.
PS D:\Desktop\RETI\LicenzeSoftware>
```

5.2.3 ClientAdmin

Aprire il terminale, navigare fino alla directory del progetto ed eseguire il seguente comando, specificando come parametri il numero della licenza:

```
python clientAdmin.py idLicenza
```

```
PS D:\Desktop\RETI\LicenzeSoftware> python clientAdmin.py LICENZA99999
Licenza LICENZA99999 invalidata con successo.
PS D:\Desktop\RETI\LicenzeSoftware>
```