Laboratorio di Programmazione di Rete - B
Laurea Triennale in Informatica. a.a.09/10
Testo del Terzo Progetto

Scadenza: 3 ottobre 2010

Andrea Corradini

Il progetto consiste nello sviluppo di un semplice sistema distribuito Client-Server per la condivisione di file.

1 Specifica generale

I client possono *pubblicare* i file in proprio possesso tramite il server, e possono *richiedere* al server dei file. Il server non fornisce un file pubblicato direttamente, ma fornisce l'indirizzo di un client che lo ha pubblicato. Sia il server che i client mantengono localmente una tabella contenente tutti i client che si sono *registrati* al server e che sono ancora *attivi*. Un client registrato è attivo se non *è uscito* volontariamente dal sistema, e se non è stato *dichiarato morto* dal server perché non ha mandato un *keep-alive* nel tempo prefissato.

Per ogni file pubblicato con successo, il server mantiene due liste di client: i *seeder* che lo posseggono, e i *leecher* che lo hanno richiesto. Un client C che vuole scaricare un file lo richiede direttamente al seeder indicato dal server in risposta alla richiesta di C. Un leecher deve notificare al server quando ha finito di scaricare un file, per essere spostato nella lista dei seeder.

Ogni client esegue una sequenza di istruzioni letta da file, che può comprendere pubblicazione di file, richiesta di file, uscita dal sistema. Sia i client che il server devono stampare sullo standard output gli eventi essenziali manifestatisi durante il loro funzionamento.

2 Requisiti

Il progetto deve soddisfare in modo puntuale i seguenti requisiti, che completano la specifica informale del precedente paragrafo.

1. I client e il server devono poter essere eseguiti sia su macchine diverse che sulla stessa macchina. Ogni client è identificato univocamente da una coppia *(InetAddress, porta)*, dove il primo è l'indirizzo dell'host, mentre *porta* è la porta su cui il client ha aperto un ServerSocket per accettare le connessioni TCP richieste da altri client. La coppia *(InetAddress, porta)* è l'*identificatore* del client.

2. Le interazioni nel sistema devono avvenire usando i seguenti protocolli:

   - UDP per la gestione dei pacchetti di keep-alive mandati dai client al server;
   - Multicast per comunicare al server e agli altri client che un client sta uscendo dal sistema;
   - TCP per la trasmissione di un file tra due client;
   - RMI per le rimanenti interazioni tra client e server.

3. Il server deve essere lanciato da linea di comando indicando come parametri ⟨porta del registro RMI⟩ (default: 1099), ⟨porta UDP⟩ del listener del keep-alive (default: 5555), e ⟨gruppo Multicast⟩ per notifica dell'uscita di un client dal sistema (default: 226.226.226.226). Si assume che il registro RMI sia attivato sullo stesso host del server.

4. Il client deve essere lanciato da linea di comando indicando i seguenti parametri, nel seguente ordine:

   - [obbligatorio] nome della working directory. In questa directory devono essere posti i file che il client pubblicherà, e si copieranno i file scaricati; inoltre deve contenere il file contenente le azioni che il client dovrà eseguire.
   - host del server (default: localhost)
   - porta del registro RMI (default: 1099)
   - porta UDP del server (default: 5555)
   - gruppo mulitcast per notifica dell'uscita di un client dal sistema (default: 226.226.226.226)

5. Ogni file nel sistema è identificato univocamente dal suo nome. Il descrittore di un file deve contenere il nome e la lunghezza in byte del file.

6. Il server deve fornire i seguenti metodi remoti:

   - `register(...)`: ha come parametri l'identificatore del client e una callback. Restituisce la lista di identificatori di client registrati in quel momento nel server. Lancia un'eccezione se il client risulta già registrato.
   - `publish(...)`: ha come parametri l'identificatore del client e un descrittore di file. Restituisce `false` se è già stato pubblicato un file con lo stesso nome, `true` altrimenti. Nel secondo caso il server inserisce il file tra quelli pubblicati, con il client che lo ha pubblicato come primo seeder.

- `search(...)`: ha come parametri l'identificatore del client e il nome di un file. Restituisce `null` se il file non è pubblicato, altrimenti restituisce una struttura contenente l'identificatore di un seeder del file e il descrittore del file. Il server inserisce il client che ha fatto la richiesta tra i leechers del file.
- `completed(...)`: ha come parametri l'identificatore del client e un descrittore di file. Segnala al server che il client ha finito di scaricare il file: il server lo sposta dalla lista dei leechers a quella dei seeders.

7. Il client deve fornire i seguenti metodi remoti (invocabili dal server tramite la callback):

   - `addClient(...)` aggiunge l'identificatore del client passato per argomento alla lista di client attivi.
   - `removeClient(...)` duale del precedente.

8. Ogni client deve inviare regolarmente un pacchetto di keep-alive al server, tramite UDP, almeno ogni 3000 msec. Se il server non riceve un pacchetto da un client per più di 3 secondi, lo considera morto: in questo caso elimina il client da tutte le liste e tramite la callback lo fa cancellare a tutti gli altri client registrati. Invocazioni RMI al server provenienti da un client che è considerato morto devono causare un'eccezione; analogamente, pacchetti keep-alive di un client considerato morto vengono ignorati.

9. Quando un client decide di uscire dal sistema, lo deve notificare al server e a tutti gli altri client attivi tramite una messaggio mulitcast, usando l'apposito gruppo multicast. Eventuali operazioni in corso al momento dell'uscita possono essere completate. In ogni caso, tutti i thread del client devono essere terminati al momento dell'uscita dal sistema.

10. Ogni client legge da un file di testo le azioni che deve eseguire. Le azioni devono essere scritte una per riga, con la seguente sintassi:

    ⟨`ACTION`⟩ `::= PUBLISH` ⟨`FILENAME`⟩ | `SEARCH` ⟨`FILENAME`⟩ | `EXIT` | `WAIT` ⟨`NUM`⟩

    dove ⟨FILENAME⟩ è una stringa e ⟨NUM⟩ un intero. Le azioni PUBLISH e SEARCH corrispondono alla pubblicazione e alla richiesta del file indicato, e vanno realizzate come indicato sopra. EXIT causa l'uscita dal sistema. WAIT ⟨NUM⟩ causa una sospensione di ⟨NUM⟩ millisecondi del solo thread che legge i comandi dal file: tutte le altre attività del client proseguono regolarmente.

11. Per poter monitorare il funzionamento del sistema, sia il server che ogni client devono scrivere sullo standard output, una per riga, tutte le operazioni salienti (e il relativo esito) dei threads che li costituiscono.

3 Modalità di consegna e valutazione

Il progetto può essere svolto da gruppi di al massimo due persone. L'orale sarà in ogni caso individuale, pertanto ogni studente sarà considerato reponsabile dell'intero progetto consegnato, anche se sviluppato in collaborazione.

La scadenza per la consengna del progetto è **Domenica 3 ottobre 2010 alle ore 24:00**: non saranno consentite proroghe. Per la consegna inviare un mail al Prof. Corradini con oggetto: [**LPR**] **Consegna progetto**, contenente relazione, manuale d'uso e sorgenti del progetto in un singolo archivio allegato. Al più tardi nei giorni immediatamente successivi deve essere consegnata una stampa di tutto il materiale presso la portineria del Dipartimento. Dopo la consegna il docente convocherà il gruppo per la discussione e l'orale. L'orale verterà sia sulla discussione del progetto che sul programma svolto durante il corso.

Il progetto può essere realizzato su qualsiasi piattaforma di sviluppo e sistema operativo, ma è necessario che sia compilabile ed eseguibile sulle macchine Linux presenti nel Polo Didattico (in particolare gli host `fujimXX` del Laboratorio M). Prima di consegnare il progetto verificare la compilazione e l'esecuzione dell'applicazione su tali host, anche utilizzando host diversi (evitare l'uso di *localhost*).