6° Esercitazione (da svolgere):

Java RMI

Utilizzando java RMI sviluppare un'applicazione C/S che consenta di effettuare le operazioni remote per:

- **contare le occorrenze di un carattere** in un file presente sul server remoto;
- **restituire la lista con i nomi di file** (presenti nel direttorio remoto) **che contengono un carattere** (indicato dal client) **nel nome stesso**.

Il progetto RMI si basa su un'interfaccia (***RemOp,*** contenuta nel file *RemOp.java*) in cui vengono definiti i metodi invocabili in remoto dal client:

- Il metodo **`conta_occorrenze`** accetta come parametro d'ingresso il nome del file e il carattere, quindi restituisce un intero corrispondente al numero di occorrenze contate; altrimenti, in caso di errore, solleva un'*eccezione remota*, ad esempio, se il file non è presente nel sistema.
- Il metodo **`lista_file_car`** accetta come parametri d'ingresso il nome del direttorio e il carattere, quindi, se il direttorio richiesto esiste, lo apre e restituisce al client la *lista dei file* il cui nome contiene il carattere richiesto; altrimenti, in caso di errore, solleva un'eccezione remota, ad esempio, se il direttorio non esiste.

Si progettino inoltre le classi:

- ***ServerImpl*** (contenuta nel file *ServerImpl.java*), che implementa i metodi del server invocabili in remoto e presenta l'interfaccia di invocazione:
  ServerImpl [registryPort]

- ***Client*** (contenuta nel file *Client.java*), che realizza l'interazione con l'utente proponendo ciclicamente i servizi che utilizzano i due metodi remoti, e stampa a video i risultati, fino alla fine del file di input da tastiera.
  Il Client presenta l'interfaccia di invocazione:
  Client NomeHost [registryPort]

NOTA
Il Registry deve essere in esecuzione su un host concordato e in ascolto alla porta eventualmente specificata.
Il server (istanza della classe relativa) deve registrare il riferimento remoto sul registry alla locazione corretta.
Il client (istanza della classe relativa) deve recuperare dal registry il riferimento all'oggetto remoto, ServerImpl, di cui deve invocare i metodi.

Proposta di estensione

Get Multiple

Si vuole sviluppare un'applicazione C/S basata su RMI e su socket con connessione per il trasferimento di tutti i file di un direttorio remoto dal server al client (multiple get). In particolare, si vogliono realizzare due modalità di trasferimento, la prima con server attivo (*il server effettua l'accept*), la seconda con client attivo (*il client effettua l'accept*). Si dovranno realizzare un **client** e un **server**; l'utente, per ogni trasferimento, decide quale delle due modalità utilizzare.

Per entrambe le modalità, si prevede un'*interazione iniziale sincrona (realizzata con una richiesta RMI sull'oggetto remoto server)* per trasferire la **lista dei file** da inviare e **l'endpoint** (host e porta) **di ascolto**; quindi, una seconda fase di *trasferimento dei file* dal server al client *(realizzata con socket connesse)*.

In particolare, per la *prima modalità*, il metodo remoto accetta come argomento di ingresso il **nome del direttorio** e restituisce una struttura dati con l'**endpoint di ascolto del server** e la **lista con i nomi e la lunghezza di tutti i file** da trasferire.

Il **client** richiede ciclicamente all'utente il nome del direttorio da trasferire ed effettua la chiamata RMI, quindi stabilisce una connessione con il server remoto e riceve i file salvandoli sul direttorio locale.

Il **server** implementa il metodo RMI richiesto ed è realizzato come server *concorrente* e *parallelo*; per ogni nuova richiesta ricevuta il processo padre, dopo aver accettato la richiesta RMI, crea la socket di ascolto, attiva un processo figlio a cui affida il completamento del servizio richiesto e restituisce la struttura dati con la lista dei file e il proprio endpoint.

Per la *seconda modalità*, il metodo remoto accetta come argomento di ingresso il **nome del direttorio** e l'**endpoint di ascolto del client** e restituisce la **lista con i nomi e la lunghezza di tutti i file** da trasferire.

Il **client** richiede ciclicamente all'utente il nome del direttorio da trasferire, crea la socket di ascolto, effettua la chiamata RMI e riceve la lista dei file da trasferire; quindi, effettua la accept e i trasferimenti di file necessari.

Il **server** implementa il metodo RMI richiesto ed è realizzato come server *concorrente* e *parallelo*; per ogni nuova richiesta ricevuta il processo padre, dopo aver accettato la richiesta RMI, crea la socket per la connessione col client, attiva un processo figlio a cui affida il completamento del servizio richiesto e restituisce la struttura dati con la lista dei file.

Consegna

Potete inviare via email l'estensione ai docenti, per discutere la soluzione ed eventualmente pubblicarla sul sito del corso