KTH

ID2212 Network Programming with Java

IT-Universitetet

Homework 1: Developing Client-Server or Peer-to-Peer Applications with Sockets

Due 2007-11-23 (Friday) -- to be reported on the lab session 2007-11-23 (Friday).
If you report a homework on 2007-11-23 (Friday) and your solution is accepted, you will get upto 3% bonus on the first exam.
If you report a homework on 2007-11-16 (Friday) and your solution is accepted, you will get upto 4% bonus on the first exam.

Objectives:

- You can develop a distributed (client/server or peer-to-peer) application, i.e. define tasks, assign tasks to processes and develop an application-specific communication protocol for process interaction using UDP or/and TCP sockets;
- You can use Java classes representing UDP and TCP sockets (the java.net package) and IO stream classes (the java.io package) to program and to use message passing in a distributed application;
    - You know how to use Socket and ServerSocket classes of the java.net package for communication between a server and clients;
    - You know how to use byte streams of the java.io package for sending/receiving protocol messages and files of different MIME types over a TCP connection such as InputStream and OutputStream, BufferedInputStream and BufferedOutputStream;
    - You know how to access files using classes the java.io package such as File, FileInputStream and FileOutputStream
    - You know how to use streams, stream and string tokenizers of java.io and java.util package for programming a character-based protocol, such as PrintWriter, BufferedReader, StreamTokenizer and StringTokenizer;
    - You know how to use object streams such as ObjectOutputStream and ObjectInputStream for sending/receiving objects over TCP-connections or in datagrams of UDP communication;
- You can develop an object-oriented Java client (or a peer) of a distributed application with a simple but yet convenient, and informative (graphical) user interface using Java AWT (or Swing) classes and interface; you can program a responsive (graphical) user interface taking into account

communication latency in a distributed application;
- You can program, create and use concurrent threads in processes (clients, peers, server) of a distributed application in order to improve scalability and performance (e.g. response time), to hide communication latency.
- You know how to evaluate performance (response time) of a client-server application; you know how to make a profile of an application.

Task:

You are to solve in Java at least one of the following problems. The homework can be done in a group of 2 students.

- Homework should be presented and demonstrated to a course instructor during a lab session on the date it is due or on the previous lab session (for extra 1% bonus on th first exam).
- **Bonus policy:** If you report a homework on time and your solution is accepted, you will get 3% bonus on the first ID2212 exam in HT07. Reporting on the precious lab session (Nov. 11) gives additional 1% bonus if your homework is accepted. The amount of bonus can be reduced for errors and/or poor design of the application.

Problem 1. A client-server guessing game (a variant of the two-player "Hangman" game)

Develop a client-server distributed application in Java for a guessing game similar to the "Hangman" game (see a description at Wikipedia.org).

The server chooses a word from a dictionary, and the client (the player) tries to guess the word chosen by the server by suggesting letters (one letter at a time) or a whole word. The client is given a certain number of attempts (failed attempts) when it may suggest a letter that does not occur in the word. If the client suggests a letter that occurs on the word, the server places the letter in all its positions in the word; otherwise the counter of allowed failed attempts is decremented. At any time the client is allowed to guess the whole word. The client wins when the client completes the word, or guesses the whole word correctly. The server wins over when the counter of allowed failed attempts reaches zero. The client and the server communicate by sending messages over a TCP connection according to the following protocol. The server should compute the total score of games using a score counter: if a client wins the score counter is incremented, if the client loses the score counter is decremented.

- If the client (player) wants to start a new game it sends a special "start game" message to the server . The server randomly chooses a word from a dictionary, sets the failed attempt counter to the number of allowed failed attempts, and replies with a message that includes a row of dashes (a current view of the word), giving the number of letters in the chosen word (e.g. "----------" for the word "programming"), and the number of allowed failed attempts (e.g. 10).
- If the client (player) wants to suggest a letter or the whole word, it sends the letter or the guessed

word to the server. The server replies as follows, depending on the word it has chosen and the client's guess.

- If the client guesses the whole word correctly, the server sends a congratulation message together with the word and the total score (see below);
- If the letter guessed by the client occurs in the word and the client has completed the entire word, the server sends a congratulation message together with the word and the total score;
- If the letter guessed by the client occurs in the word and the has not completed yet the whole word, the server sends to the client the current view of the word with the letter placed in all its correct positions and the current value of the failed attempts counter. For example, if the client suggest "g" then the current view of the word is "---g------g", if the client then suggest "m" then the current view of the work becomes "---g--mm--g";
- If the letter guessed by the client does not occur in the word or the client guesses the whole word incorrectly, the server decrements the failed attempt counter and, depending on the counter's value, sends either the current view of the word together with the value of the failed attempt counter (if the counter > 0), or a "gave over; you loose" message together with the total score (if the counter = 0).

The client G(UI) should display a current view of the word, the current value of the failed attempts counter, and the current value of the total score.
As a source of words for the server, the server can use an online dictionary in `/usr/dict/words` under Solaris or Linux. The file is also available at http://www.imit.kth.se/courses/ID2212/homeworks/words. The file contains 25,143 words (and is used by the Unix `spell` command).

Problem 2. A Rock/Scissors/Paper Game

Develop a distributed peer-to-peer application for a three-player rock/scissors/paper game. Each player runs a Java application (a peer) with a (graphical) user interface that allows to play the game and show scores. Using (G)UI, each player chooses one of rock, scissors, or paper. Then the players (peers) communicate their choices to each other in order to compare the choices to see who "won." Rock smashes scissors, scissors cut paper, and paper covers rock. Award a player 2 points if it beats both the others; award two players 1 point each if they both beat the third; otherwise award no points. Then the players play another game. After each game (G)UI shows the score of the game and the total scores. You do not need to provide an advanced and complicated (G)UI, try to make it simple.

Use one peer process for each player; do not use an additional coordinator process. The peers must communicate with each other using either datagram (UDP) sockets or/and multicast sockets or TCP sockets (see Socket and ServerSocket classes).

Optionally, you can implement this game as a set of applets interacting with each other via a relay server that resides on the host from which the applets were downloaded (i.e. the host that runs the Web server). The applets can open and use a TCP connection to the server.

Problem 3. The Common Meeting Time Problem

Develop a distributed peer-to-peer application that allows ***n*** persons to determine common meeting times at which they can meet. Each person creates a file with a set of records being times (sorted in ascending order) at which s/he is available for the meeting. Then the person starts a Java application (a peer) that reads the file and interacts with other peers until it either determines all possible common meeting times or determines there are no common meeting times, i.e. it is not possible to appoint a meeting at times specified. The peer should be given the number of persons (***n***) as an input parameter. Use one peer process for each person; do not use an additional coordinator process. The peers must communicate with each other using either datagram (UDP) sockets or/and multicast sockets or TCP sockets (see Socket and ServerSocket classes).

Links

- Lecture 4. Networking with Sockets
  - Exercise 1.Design of client-server applications:
  - Lab 1. Socket Communications. Clients and Servers
- Lecture 3. GUI Programming. Multithreading with Java
- Java Plafrom SE Documentation
- Java Networking
  - Trail: Custom Networking (in Java Tutorial)
  - Networking Features (in Java SE, documentation)
- Creating a GUI with JFC/Swing
- Multithreading in Java:
  - Lesson: Concurrency (in Java Tutorial)
  - Concurrency Utilities

Latest update: November 8, 2007
2G1529-teachers(a)it.kth.se