CS181, Programming Languages

Assignment 6

(due: 11-27-2000)

Problem one:

Create a multi-threaded server that listens on a pre-defined port for client requests.

- **Server specification**

  The server should run as a separate thread, receiving client packets and storing them in a queue for further processing. Use a *DatagramSocket* for receiving packets sent by the clients (UDP protocol.) The server should use the *ThreadPool* class (from the previous assignment) for storing and executing client requests. Create the *Parser* inner class that implements the *Runnable* interface, parses the packets and executes the appropriate task. Notice that by using the *ThreadPool* class you don't have to worry about creating, starting and stopping threads. You only have to create a **parse** function that parses packets according to the specification (a simple byte would suffice in this case), call it from the **run** method of *Parser* inner class and complete the request! For every new packet that arrives, just create a new Parser instance associated with the packet and add it for execution in the *ThreadPool*. The server should run on **hill.cs.ucr.edu**. The server port should be defined by you and should be an integer greater than 2000 (if somebody else is using the same port at the same time, the *DatagramSocket* creation will fail.)

- **Client specification**

  Create a simple GUI that will act as the client application. Use a combobox for selecting the type of service and a label that will print the answer. Use a button for sending the request to the server. The client should create a datagram packet containing the appropriate data and use a *DatagramSocket* to send the packet to the server (running on hill), using the pre-defined port. Than it should wait for a reply containing the answer. Notice that if any packet gets lost, the client will wait forever! Don't worry about reliable communication (at the time!) Assume that no packets will get lost.

- **Client requests**

  The server should be able to handle the following requests:

1. Send the local time as a string.
2. Send the name of the thread that is handling this request (Use the Thread.currentThread() method to get a reference to the currently executing thread.)
3. Send the number of active threads on the server

You might have to modify your *ThreadPool* code or you could use the solution that will be posted pretty soon!