**ECET-490**

**UDP Programming Project**

**Journaling Application**

This week, we have been looking at using Java UDP Sockets to build networked applications. The example provided with this week’s lecture demonstrates some of the basic ideas for setting up a networked application using UDP in Java. This week’s programming assignment is to design and build a journal application using UDP.

The journaling application is to be designed so that a client can access their journal entries from anywhere. They can store any text information in their journal. This journaling system should be set up to support the following operations:

1. The client’s user name is used by the server to locate journal entries made by that client.
2. When the client starts the journal application, he or she must identify themselves with their user name and indicate which journal entry they want to work on. Journal entries are identified by the month, day, and year. There is only one journal entry per month-day-year per client.
3. A client can access the journal entry from any date. For example, a client may create a new journal entry in the morning. Later in the day, the client may continue working on the journal entry for that day. This means, the server retrieves the journal entry and sends it to the client where is it displayed in a text area. The client can change the entry or add to it. When finished, the client can once again save the entry, which sends the updated text to the server for storage.
4. From the above operations, the journal server must support reading a journal entry for a specific client user name and month-day-year. It must also support saving a journal entry for a specific client user name and month-day-year.

The primary requirement for this lab is that you must use UDP for communications between the client and the server. All client journal entries must be saved to files. I am providing you with a Java class called: the JournalManager, which is designed to read and write journal entries to and from files. You should design your journal server to make use of this JournalManager class.

You should spend some time thinking about how to design this application, with particular focus on the application protocol you will need to develop to support this application. Refer to the Application Protocol Design document provided with this week’s material. What can a client request? What replies can the server generate? What errors could occur that a client needs to know about? What information needs to be sent from the client to the server; from the server to the client? What is the maximum message size your client and server will be able to exchange? What is the structure of each message exchanged? How are you going to implement a structured message with a Datagram Packet? (Hint: I would suggest looking at the StringTokenizer class that is designed to break a String into a set of substrings.) What about synchronization? When will the client wait and for how long? When will the server wait and for how long?

Remember, at this point, we are still basically working with a single threaded server, so while the server is servicing one client request, all other client requests are waiting.

For the first part of the lab, I want you to capture the design of your message exchange in a written protocol specification. Be very specific about the structure of messages that will be exchanged. Define what values you are going to use as part of your message exchange. Fully define the message exchange scenarios. For instance, if the client sends message A, the server can respond with message B or C depending on how the server handles A. Include any possible error responses that could come back from the server. The protocol write-up should take no more than two pages. Include an event diagram that illustrates the message exchanges and the synchronization between clients and the server.

**Once you have defined your protocol, you can proceed to implementing your program. Demo your program to the professor, get a sign-off, and turn in screen shots of the client and server at work, and print-outs of the source code for both the client and server. Also turn in your written protocol specification.**

**import** java.io.\*;

**import** java.util.\*;

**public** **class** JournalManager

{

 String baseDirectory;

 **public** JournalManager(String basedir)

 {

 baseDirectory = basedir;

 File dir = **new** File(basedir);

 **if** (dir.exists() == **false**)

 {

 dir.mkdir();

 }

 }

 **public** String readFile(String fromClient, String month, String day, String year)

 {

 File journalEntry;

 String result;

 journalEntry = **new** File(baseDirectory + "\\" + fromClient + "\\"

 + month + "." + day + "." + year);

 **if** (journalEntry.exists() == **false**)

 {

 **return** "Requested journal entry not found!";

 }

 **else**

 {

 **try**

 {

 Scanner in = **new** Scanner(journalEntry);

 result = in.nextLine();

 **while**(in.hasNextLine())

 {

 result += "\n" + in.nextLine();

 }

 in.close();

 }

 **catch**(FileNotFoundException fnf)

 {

 result = "Requested journal entry not found";

 }

 }

 **return** result;

 }

 **public** String writeFile(String fromClient, String month, String day, String year, String entry)

 {

 File journalEntry;

 File clientDir = **new** File(baseDirectory + "\\" + fromClient);

 **if** (clientDir.exists() == **false**)

 {

 clientDir.mkdir();

 }

 journalEntry = **new** File(baseDirectory + "\\" + fromClient + "\\" +

 month + "." + day + "." + year);

 **try**

 {

 Formatter out = **new** Formatter(journalEntry);

 out.format("%s",entry);

 out.close();

 **return** "Message received and stored.";

 }

 **catch**(FileNotFoundException fnf)

 {

 **return** "Server error occurred while trying to store message.";

 }

 }

}