420/740 Programming Assignment 1

Due Date: Saturday December 8, 2012 at 11:59:00pm

Late projects will not be accepted and will result in a 0

Overview

Most operating systems provide the ability for both system and user processes to log messages in a log file. The messages in a log file can be used by a system administrator to monitor the activities of a system, to look for potential problems, and to troubleshoot problems when they occur. In most systems the log files are stored on a per machine basis, which means an administrator has to potentially look through log files on dozens of machines when trying to track down a problem. Another solution to this problem is to provide one central logging facility that is used by all the machines on a network. In this assignment you will write both the client and server code to implement a distributed system log facility.

System Organization

The distributed log system consists of a server, typically a single server, and several clients that use this server to record log messages. Before a client can log a message on a log server it must obtain a ticket. Tickets are used by the server to authenticate the clients and to provide the clients with the ability to separate their log messages into groups. For example, a distributed course registration system may want to separate the log messages that deal with problems with course data, from network problems. The course registration system could do this by obtaining two tickets from the log server: one would be used to log course data problems, whereas the other could be used to record network related messages.

The log service also provides a client with the ability to retrieve all of the messages associated with a given ticket. This capability might be used by an administrator to read log messages remotely. The server maintains the log messages for a given ticket until the ticket is released by a client. When a ticket is released, all of the messages associated with the ticket are discarded and the ticket is considered invalid (i.e. clients can no longer log messages using the ticket).

The Protocol

The log service uses TCP to transmit messages between clients and the server. Messages consist of a series of characters terminated by a newline character. The first character in every message sent by a client to a server is a character that identifies the type of request being made by the client. The remaining information in the message depends upon the type of request that is being made. The four different request types, and the information included in the message, are summarized in the table below:

Request	**First Character**	**Additional Information**	**Response**
New Ticket	0	*none*	*ticket*
Log a Message	1	*ticket:message*	none
Release a Ticket	2	*ticket*	none
Get Messages	3	*ticket*	*count* followed by *message* (each *message* is on a new line)

For example, if I send *0* to the server, it will respond with a ticket number (ie. ABCD1234). I can then use that ticket number to log a message with the message: *1ABCD1234:hello world*

If the server receives a malformed request, or the request cannot be carried out, the request will simply be ignored by the server.

Part I

You are to write a class named **LogClient** that implements the **ILogService** (**LogService** in Java) interface. This class can be used by a client to access the distributed logging system. All of the details regarding communication with the log server are encapsulated in the LogClient class. I have written a very simple command line driven client in c# that illustrates how the ILogService interface can be used to access the distributed log service. This is accessible in the Program.cs file. You can obtain the source code for the Interface in Java and c#, as well as the simple client from here.

For the duration of this assignment you should be able to access my distributed log server that is running on kayrun.cs.rit.edu. My version of the server maintains two log files: LogServer.debug that contains debugging information, and LogServer.log that contains a permanent record of all log entries (even if the ticket has been released). These files are accessible on my web site (click the links above to see them). The server is running on port 6007. The first line in the LogServer.debug file will always specify the port number the server is currently using, in the event that it changes. If the service is not running where you think it should be, check this file before contacting me.

When you are working on developing your client, or simply using telnet to explore the protocol used in this application, the LogServer.debug and LogServer.log may help you to understand what is going on when you try communicating with the service. Note that the information in these files are open to the public **do not put any messages in these files that people may find offensive or that contains personal information**.

Part II

Part II Write a server for the distributed logging service. Your server must implement the exact same protocol that my server implements. I will test your server by running a test client that I have written against your running server. Your server should record all log messages to a file named LogServer.log. You must also log debugging information in a separate file as well. The exact format of the file(s), and their contents are left up to you to decide. Minimally, the file containing the log messages must contain every message logged by every client and the ticket number that was used to log the message. You might find it useful to look at the log files created by my server for guidance. **Note that the messages associated with an open ticket should be stored in memory for quick retrieval (in other words the file that contains copies of the log messages is for archival purposes only).**

The server that you write does not have to use multiple threads (you can use multiple threads if you wish). You should not need to use the interface to generate the Server. By default, your server should be running on port 6007.

The easiest way to generate a ticket is to use a UUID or GUID. In Java 1.5, the UUID class is available in Java.Util.UUID. In c#, the GUID is available in the System.GUID class.

Style

Write clean, modular, well-structured, and well-documented code. Here are some general guidelines:

1. Make sure that your name appears somewhere in each source file.
2. Each class and every method in the class should have a header that describes what the class/method does.
3. You do not have to document every single line of code you write. Provide enough documentation so someone who is not familiar with your program can figure it out
4. Names of classes, method, constants, and variables should be indicative of their purpose.
5. All literal constants other than trivial ones (e.g. 0 and 1) should be defined symbolically.

Submitting

Place all your source files in a single directory and zip the files up (in zip format). Submit the archive to myCourses, in the project 1 folder.

You may include a README file in this directory, if you wish, but no other files are acceptable.