|  |
| --- |
| Marist College |
| Server Client Network Programming in Java |
| **Final WriteUp**  Professor: **Juan Arias**  **By-**  **Prabuddha Banerjee** |

Prabuddha Banerjee 11/22/2019

**Final Milestone**

This Project is dealing with Networking as per the proposal. In this we deal with sending of Audio files via Server and client socket network. I wanted to make a communication system in which the messages are communicated among clients securely.

I decided to do this networking project because I am very much motivated in learning new things during my course work of Masters and this would be very good opportunity to learn new thing under the guidance of professor.

This project is focused on implementing an infrastructure that allows the distributed program components to communicate over a network in a reliable, efficient and generic way. The goal of the mechanism is to hide the distributed nature of remote objects as well as maintain privacy of the files which are sent. In particular, this project will communicate between clients via server network port and in order of communication it will maintain the privacy of message by encoding the file.

The system deals with maintaining transparency which is an important parameter in software design, it could simplify the end user’s way of sending messages securely. In this project up till now I have created methods to encrypt the sound files and to decrypt it. Further I have also built a method to send messages between client and server.

Before Starting to understand the how the Client- Server Socket we need to understand what is a socket? A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

So first Class that I dealt was the Server Class and Client class

Server Class

This is the class that provides a system-independent implementation of the server side of a client/server socket connection. The constructor for ServerSocket throws an exception if it can't listen on the specified port (for example, the port is already being used) i.e. an IOException.

If the server successfully binds to its port, then the ServerSocket object is successfully created and the server continues to the next step—accepting a connection from a client (the next statement in the try-with-resources statement):

**clientSocket = serverSocket.accept();**

Then I sent the WAV file using the AudioInputStream and by reading it one by one each line.

Client Class

This class implements the protocol that the client and server use to communicate. This class keeps track of where the client and the server are in their conversation and serves up the server's response to the client's statements. The class implements the client program that speaks to the Server Class.

In this class usually we use the while loop that implements the communication between the client and the server. The server speaks first, so the client must listen first. The client does this by reading from the input stream attached to the socket.

During my Milestone I came across in this part in which I was earlier using FileOutputStream and ByteArrayOutput stream to fetch the WAV file from the Server side. The whole motive was to fetch the file from server and play the modulated version in client side. But I was not able to read the file from the client side. So later on I took that the help from Professor Arias who guided me to change the method of reading of my file which is sent from Server side.

Further following is the list of classes that are used for modulation and rest other purposes as explained underneath for the project:

1. SimpleSoundPlayer Class

This class deals with conversion of the sound wav file and then conversion into bytes type for the purpose of reading of the WAV file. So, firstly the sound file is in InputStream format when present in the driver. Later, when the SimpleSoundPlayer Class is called it passes the InputStream formatted file and then converts the file into the bytes. Once the file stream is passed to SimpleSoundPlayer Class, its constructor opens the file then later on the methods getSamples() and getSamples(InputStream audioStream) are used to convert the files in array of bytes.

Hence, this Class deals with just opening of the Sound(WAV) file and then playing.

1. SoundFilter Class

This is an Abstract Class designed to filter sound samples. Since SoundFilters may use internal buffering of samples, a new SoundFilter object is created for every sound played. However, SoundFilters can be reused after they are finished by called reset method.

When we call this class it assumes that all samples are of 16 bit, signed, little endian format.

This class has multiple methods which are as follows:

1. getRemainingSize():

This method fetches the remaining size in bytes after playing the sound. An example would be when we play an echoed sound for modulation the it plays longer than its actual sound file. The method returns 0 by default.

1. Abstract Filter():

This an abstract class for the implementation of filters. It filters an array of samples, provided the samples are in 16 bit and signed, little-endian format. This method is implemented by child classes

1. getSamples():

This method is created to convert a byte array to 16- bit sample. Its like other getter methods in which we pass the array and get the 16 bit in return.

1. setSample():

It’s a setter method for setting a 16 bit sample in the byte arrays. Note that the setting to byte array must be done by a 16 bit sample only.

1. EchoFilter Class

This class is a sound filter class that emulates the echo. It basically extends the SoundFilter class (which is an abstract class)

1. EchoFilter() Constructor:

It creates an EchoFilter with the specified number of delay of samples and specified decayed rate.

The number of delay samples specifies how long before the echo is initially heard. For 1 second echo with mono, 44100 Hz sound delay samples.

The delay value is how much the echo has decayed from the source. A decay value 0.5 means the echo heard is half as loud as the source.

1. getRemainingSize():

It gets the remaining size in bytes, that this filter can echo after the sound is done playing. It ensures the sound will have decayed to below 1% of maximum volume i.e. the amplitude.

1. Reset():

It clears the EchoFilter’s internal delay buffer.

1. Filter():

This method filters the sound to add an echo. The sample played are added to the sound in the delay buffer multiplied by the decayed rate. The result is then stored in the delayed buffer, so multiplied echoes are heard.

1. FilteredSoundStream Class

FilteredSoundStream class is a child class of FilterInputStream that applies a SoundFilter to the underlying input stream.

This class is already available in java docs which is inturn extends the InputStream Class.

1. FilteredSoundStream():

It’s a parameterized constructor with specified InputStream so it calls the super constructor and the SoundFilter.

1. Read():

It is the override method of the FilterInputStream to apply filter whenever bytes are read.

In this method the sound samples are read and filtered first using soundFilter’s filter method. Thereafter, if there are no remains bytes of the sound sample then we check whether there are any remaining bytes i.e. the echo.

**Below is the UML which I used for the whole project:**

|  |
| --- |
| SimpleSoundPlayer |
| -format:AudioFormat  -samples:byte[] |
| +SimpleSoundPlayer(filename:String)  +getSamples():byte[]  + getSamples(audioStream: InputStream):byte[]  +play(source:InputStream) |

|  |
| --- |
| FilteredSoundStream |
| -format:AudioFormat  -samples:byte[] |
| +SimpleSoundPlayer(filename:String)  +getSamples():byte[]  + getSamples(audioStream:AudioInputStream):byte[]  +play(source:InputStream) |
| EchoFilter |
| -format:AudioFormat  -samples:byte[] |
| +SimpleSoundPlayer(filename:String)  +getSamples():byte[]  + getSamples(audioStream:AudioInputStream):byte[]  +play(source:InputStream) |

|  |
| --- |
| SoundFilter |
| -format:AudioFormat  -samples:byte[] |
| +SimpleSoundPlayer(filename:String)  +getSamples():byte[]  + getSamples(audioStream:AudioInputStream):byte[]  +play(source:InputStream) |

|  |
| --- |
| Server |
| -connection:Socket  -server:ServerSocket |
| +Server (port:int)  +run()  +saveFile (cs:Socket) |

|  |
| --- |
| Client |
| -s:Socket |
| +Client (host: String, port: int, file: String)  + sendFile (file: String) |

Requirements

The system is dealing with the modernizing communication facilities that we could use for maintaining privacy among people. For the user to use the system one must have sound files in WAV form. The program will not work for any other format of music.

Literature Survey:

The other websites like java-forums.org have dealt with sending just sound files between clients and servers but they either seem to not cover enough or seem to focus on the sending sound file aspect too much to actually teach anything, and the system lack to be secured while sending/communicating between utilities.

User Manual:

For starting up with the system user must have any sound file in computer with WAV format. Then the user need build a connection between server and client. So firstly the user needs to run server class and then the client class to build the connection. Then the user may give the path for the file where the voice/ sound file is stored. The user may see that the file got modulated (or encrypted in our case). Then the file will be sent back to client in decrypted way.

Conclusion:

The way this program should be used is one must have WAV file at the beginning. Then the user must know what the location of the sound file is. The user will then build connection and the file is communicated. The project is not fully functional in its last stage as up till now a connection is built between server and client and the file which needs to be transferred is modulated and ready to be sent. So, just a method that needs to be set up for reading of the file via server and client network has some error. Rest of the file is fully tested and is completed by help and support of professor Arias.

***References:***

[**https://stackoverflow.com/questions/17044644/sending-audio-stream-over-tcp-unsupportedaudiofileexception**](https://stackoverflow.com/questions/17044644/sending-audio-stream-over-tcp-unsupportedaudiofileexception)

[**https://www.youtube.com/watch?v=-xKgxqG411c**](https://www.youtube.com/watch?v=-xKgxqG411c)

*Introduction to Java Programming and Data Structure by Y. Daniel Liang*