Chris Klassen – Melvin Loho – Jonathan Chu – Joel Mabbott

BCIT  3700 Willingdon Avenue

Windows assignment 4 design document

Table of Contents

[Requirements 2](#_Toc411875591)

[State Flow Diagrams 3](#_Toc411875592)

[Server Pseudo-Code 4](#_Toc411875593)

[Client Pseudo-Code 4](#_Toc411875594)

# Requirements

* Create a Windows application
* Use the TCP/IP protocol suite to transfer audio from a server to clients
* Play audio from the clients
* Client must be able to connect to a remote server
* Client must be able to save and retrieve sound files from the server
* The .wav format must be supported
* Two-way microphone support must be provided between clients
* The server must use multicasting and unicasting

# State Flow Diagrams

# Server Pseudo-Code

## Initialization

Main Function

{

Call the Setup Function

Call the Load Tracklist Function

Call the Open Socket Listener Function

If the mode is multicast

{

Call the Multicast Function

}

Else

{

Call the Unicast Function

}

Call the Teardown Function

If the Teardown Function fails

{

Print error message

exit

}

exit

}

Setup Function

{

Parse the command line arguments

If the command line arguments are invalid

{

Print the usage string

Return false

}

Set the mode to argument 1

Set the port to argument 2

Start Winsock

Return true

}

Load Tracklist Function

{

If the music folder does not exist

{

Print error message

Return false

}

If the number of audio tracks is zero

{

Print error message

Return false

}

For each audio track in the music folder

{

Add the file name to a music list

}

Return true

}

Open Socket Listener Function

{

Open a listener socket with the specified port and protocol

If the socket fails to open

{

Print error message

Return false

}

Bind the listener socket to an available port

If the bind fails

{

Print error message

Return false

}

Return true

}

Control Message Handler Function

{

}

## Multicast

Multicast Function

{

Open a multicast socket

Connect the multicast socket to a multicast group

Call the Play Music Function

}

Play Music Function

{

While the done signal has not been received

{

Pick a random song from the tracklist

Create a new thread to send clients the current song

Open the song file

While we have not reached the end of the file and

the done signal has not been received

{

Send 512 bytes of the song to the multicast socket

}

If the done signal has not been received

{

Call the Load Tracklist Function

}

}

}

## Unicast

Unicast Function

{

While the done signal has not been received

{

Do nothing

}

}

## Teardown

Teardown Function

{

Close the control channel

If the control channel fails to close

{

Print an error message

Return false

}

Stop Winsock

Return true

}

# Client Pseudo-Code

## Initialize

# Control Message Protocol

## Overview

The server and its clients communicate their intents and status changes using a TCP control channel. This control channel is designed in such a way that any number of new events can be added with relative ease.

The control channel works via message strings that can be parsed into a message structure. From this point, the structure can be interpreted and the data can be handled appropriately.

## Message Structure

Every message in this environment has a specific structure. This structure is as follows:

Message

{

Message Type

Message Data (vector of strings)

}

This structure is converted into a string to send through the control channel. The string is defined as follows:

“Message Type~Message Data 1`Message Data 2`…”

This string is delimited by ~ characters. Spaces are considered to be part of the data, but should only appear in the Message Data. Message data can be segmented using ` characters. These will be separated in the vector.

All control messages are variable length and should be read until there is nothing left.

## Types of Messages

Messages sent between the server and its clients can exist in a finite number of message types. These types are defined and described below:

|  |  |  |
| --- | --- | --- |
| MESSAGE\_TYPE | MESSAGE\_DATA | Description |
| END\_CONNECTION | Empty | This message is sent by either the server or client to indicate that the channel should be torn down. |
| START\_CONNECTION | Server Mode | This message is sent by the server to a new client to indicate what mode the server is in. |
| TRACK\_LIST | Song 1`Song 2`… | This message is sent by the server to clients when a client connects, and when a song ends. It contains a list of all server songs. |
| PLAY\_SONG | Song Name | This message is sent by the client to the server to request a song to be played. |
| SAVE\_SONG | Song Name | This message is sent by the client to the server to request a song to be saved. |
| NOW\_PLAYING | Song Name`Artist`Album`Length | This message is sent by the server to clients to indicate the next song to be played (unicast and multicast) |
| SONG\_PIECE | Song Data | This message is sent by the server to a client. It contains some data to be saved to a file. |
| END\_SONG | Empty | This message is sent by the server to a client when a song has finished streaming or being sent. |