Jack McGuire – 100249558

Assignment 2 - Streaming Music Player and Server

Wayne Rippin - 6CC514

CONTENTS

1. Evidence Of Testing

* Separate Machines
* Durability

1. READ ME

* Instructions
* Controls

1. Evidence of Testing

This section describes the tools and techniques used to analyse the functionality and performance of my Audio Streaming client and server applications.

Separate Machines

In order to determine that both the client and server applications, could be capable of running on  
multiple machines, I altered the code generation properties of both projects, so that each application’s RELEASE, executable files would be deployed with the run time library of Multi-Threaded.

This means that the client and server programs are capable of being run as standalone executables without requiring specific third party Visual Studio dependencies to be pre-installed, i.e. Dynamic Link Libraries.

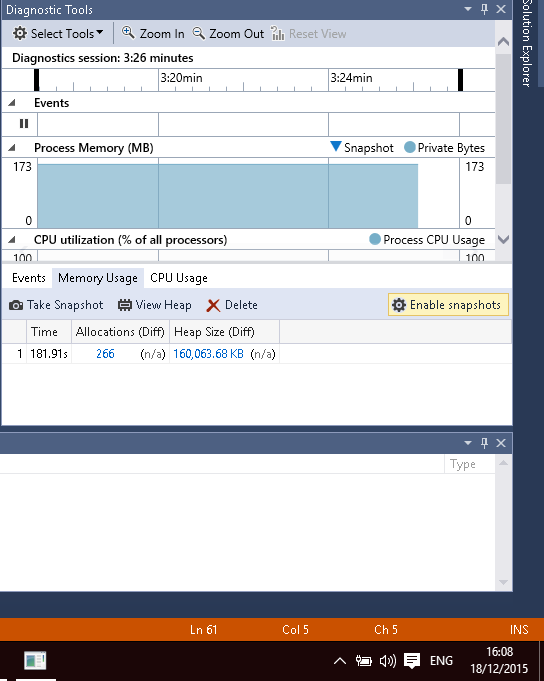
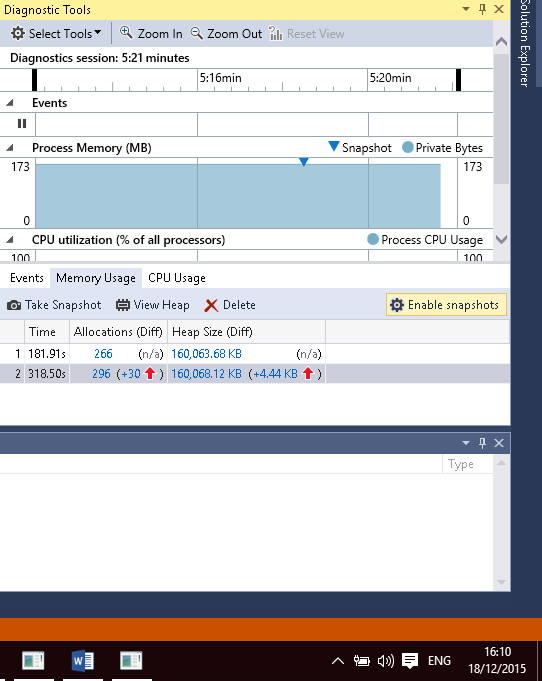
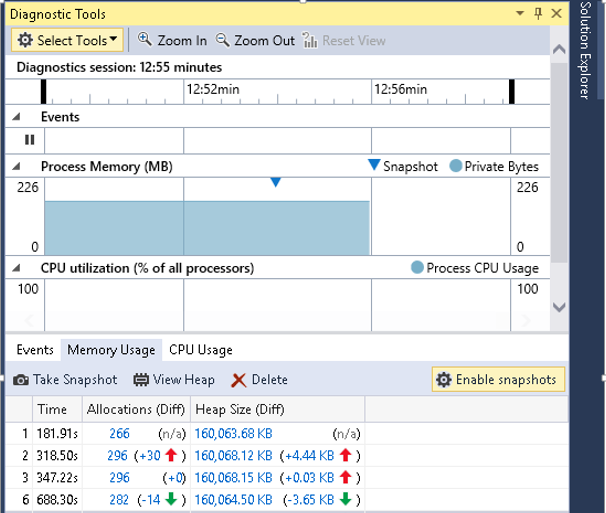
By doing this, this allowed me to distribute the applications to multiple systems with ease.

Using two of my own systems and a friend’s computer running on a different internet connection.  
I conducted tests to see whether my applications were capable of running on separate machines.

I firstly started out by setting up one of the two systems on my Local Area Network to be the host/server of my audio streaming service, while the other, would be used to run the client.  
This required that I had to configure the client’s connection settings, to use the host machine’s local IP address assigned by a router, to test the ability whether my client and server applications would work on independent machines, by communicating over a local area network.  
  
I then went on to alter the firewall settings of my router to expose a port number on the host machine, to allow my friend’s computer running an instance of my client application to communicate with my server over the internet.

However this produced Interesting results, this is because once the client on my friend’s computer attempted to select a song for playback, the application would crash unexpectedly, upon further investigation latency was causing the issue, as chunks of data were being delivered at a slower pace then what the client application was expecting, resulting in objects not being converted/serialized properly resulting in errors and crashing behaviour.

In order to remedy this, I improved the client application to specifically know how much data to look for at any given time, meaning large amounts of data to clump up unexpectedly will unlikely to occur in future because the application will not continue until a specific amount of data and no larger has been transferred.



Durability

In order to test both client and server running for obvious memory leaks and application performance, over a substantial amount of time, I used the following tools and techniques.

Using the inbuilt diagnostics tool in Visual Studio 2015, I was able to monitor the managed and native heap of both applications’ memory over a 12 hour period.

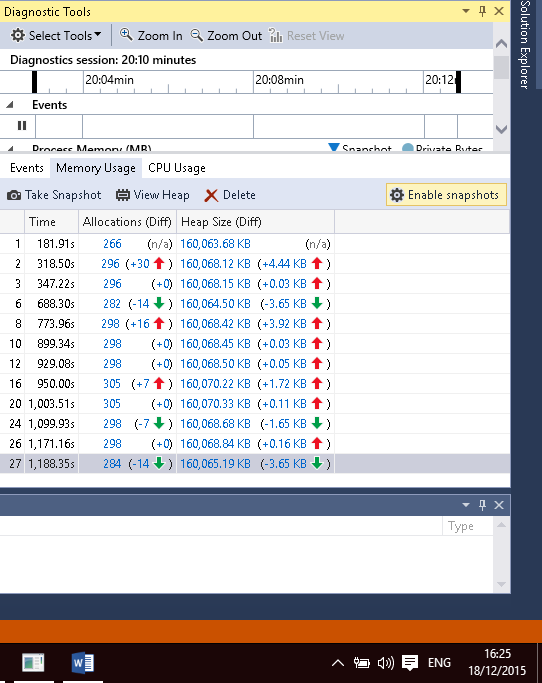
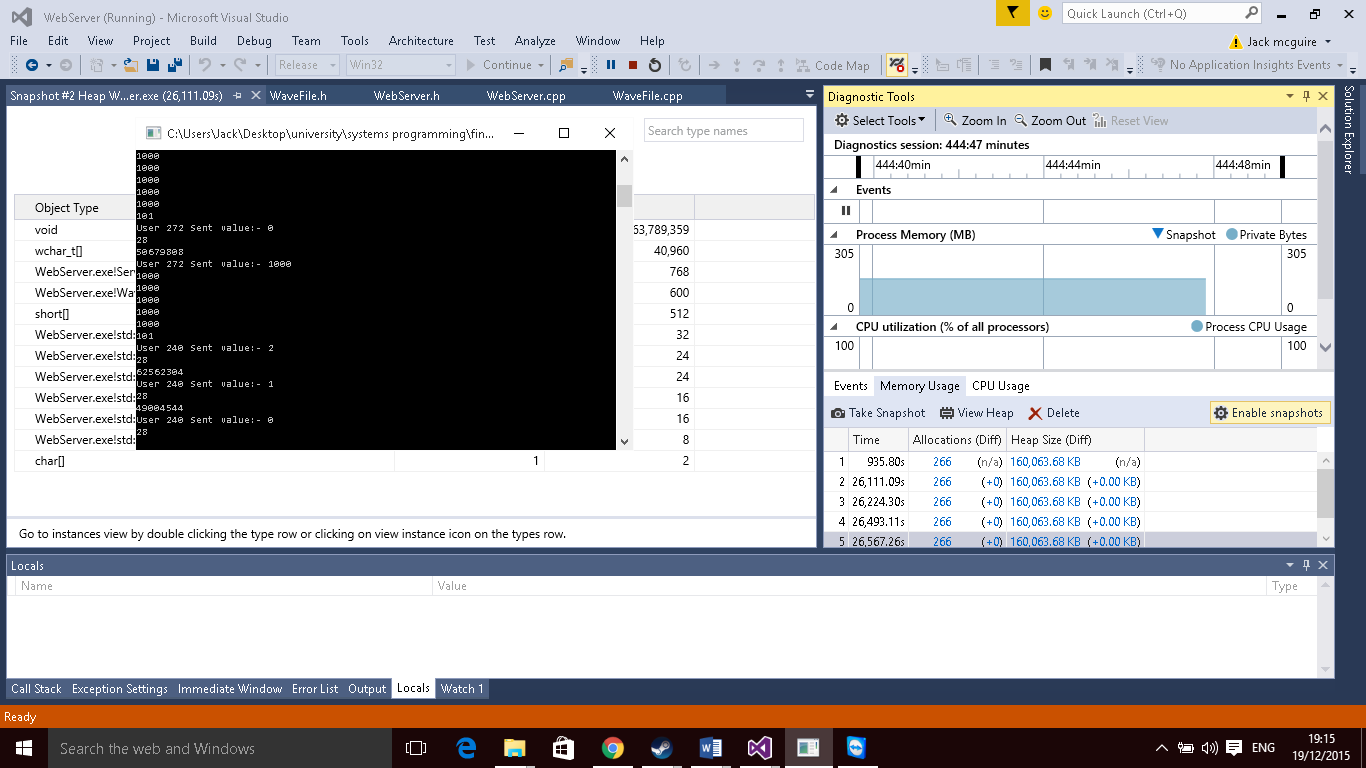
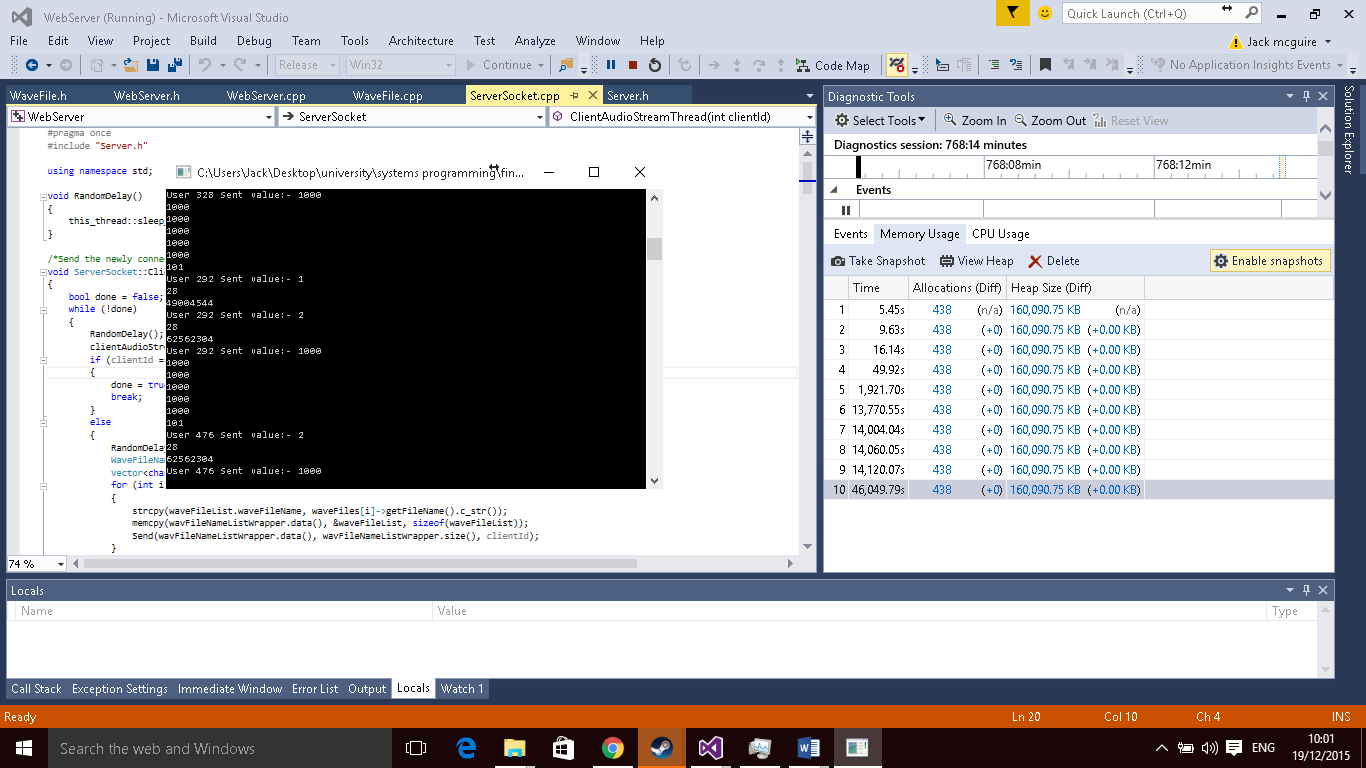
I recorded changes in memory usage by taking frequent snapshots, this provided statistics of memory use by comparing the increase and deallocation of memory to other snapshots taken earlier, using this tool gave me the means of being able to identify potential or obvious memory leaks.

Starting my application, the initial memory allocated size is 160,063.68KB, this is because several audio files have been initialized and ready for future use.

After running the server for 5 minutes, I introduced a client to connect to the server. As you can see the memory allocation was increased by 4.44kb, upon further investigation this was an increase primarily caused by the introduction of new threads represented as void objects.

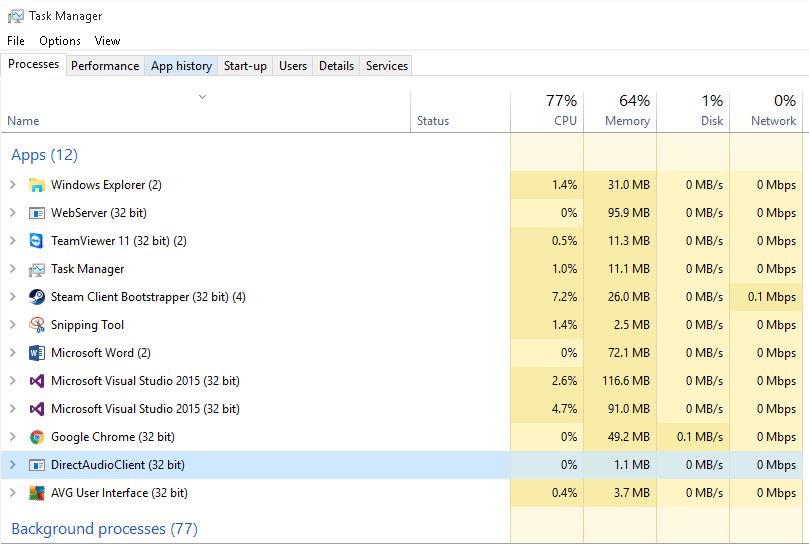
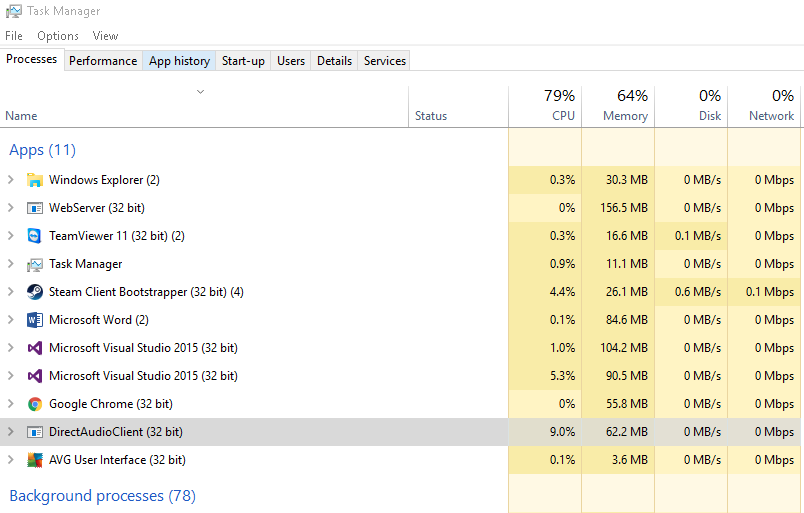
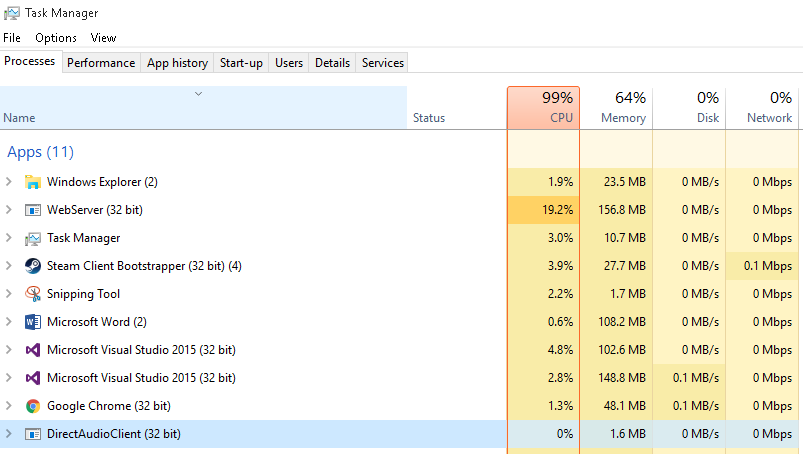
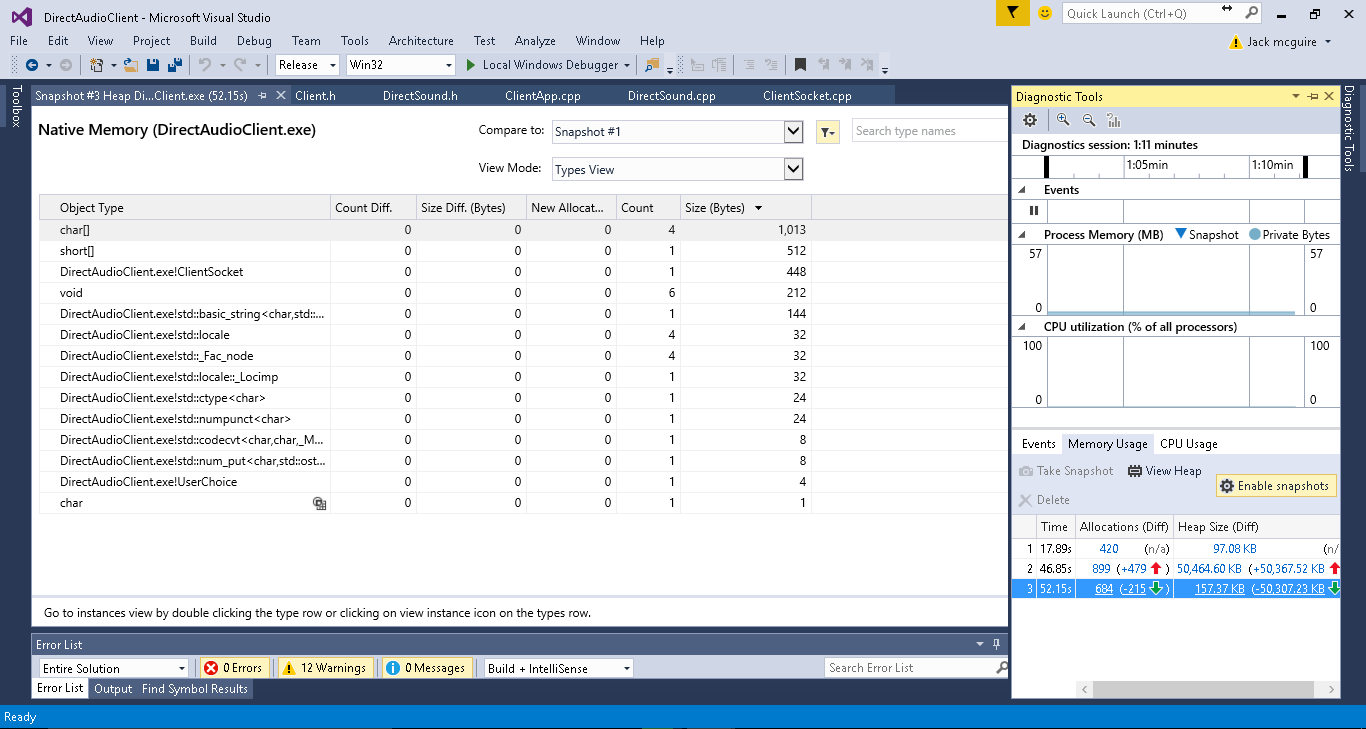
After using the client to play multiple songs, I then disconnected the client from the server application, which the server can be seen deallocating resources, to equal slightly more than the original resources allocated when the client originally first joined.

I then continued to capture additional snapshots every few hours over the period of the 12 hours.  
The image below, shows the allocation of memory after 20 minutes of run time, after multiple clients have been connecting and disconnecting, the snapshots taken display that the memory allocation returned to slightly more then what was first allocated, but removed the majority of allocations assigned  
when a client makes a new connection.



This final image displays the screen capture of the server having being run for just over 12 hours, again with numerous amounts of clients having been run, the continued to display no obvious memory leaks, further investigation of the snapshots displayed that the majority of memory allocations were mostly void objects or threads, meaning most data contains, i.e. char arrays, etc. were being efficiently managed.

The image below shows a screen capture of the snapshots taken within 7 hours into the testing period, the performance of the server application does not seem to be have been impaired as the memory allocation has not phenomenally increased with no signs of obvious resources leakages.



Below the client finished playing the song, although there is a dramatic decrease in memory usage there appears to be a 0.6 increase in comparison to when the application first started then once had finished playing.  
Upon further investigation using the diagnostics tool, I found no differences of the native heap of allocations between the before and after the song had played snapshots.

Looking below you can see the increase of CPU usage and a increased memory size, this is purely because of the playing functionality and the ever increasing third buffer used to store the audio data of a song which will continuously increase as my data is being streamed / pulled from the server.

I used task manager to capture CPU and Memory usage before a song was played, during and after a song had finished to have a better understanding of usage demands, on the client side of the application.

As you can see the client is running with 0% of CPU usage with a 1.1mb memory usage already.

**READ ME**

In order to use my Wave File streaming Server and client, the following steps have to be taken.

**INSTRUCTIONS**

1. Open ServerSettings.ini

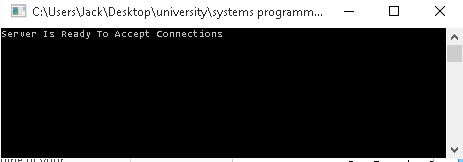
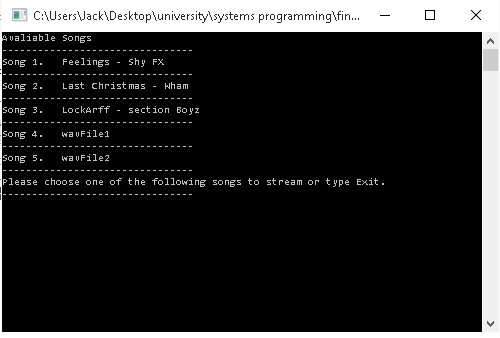
* Set Hostname IP to ‘localhost’ or the host machine’s Local Area Network IP address
* Set Desired Port Number to listen on
* Set PATH to the directory of the folder where .wav files are located

1. Open ClientSettings.ini

* Set appropriate host name IPv4 Address
* Set appropriate host port number

1. Run the Server application
2. Run the Client application  
   - choose a song for playback  
   - use controls listed below to control playback

|  |  |
| --- | --- |
| Command | Result |
| http://www.wpclipart.com/computer/keyboard_keys/large_keys/computer_key_Enter.jpg | ENTER/ RETURN will force playback to stop entirely |
| http://www.wpclipart.com/computer/keyboard_keys/large_keys/computer_key_Space_bar.png | SPACE BAR allows the user to pause and resume playback |
| http://cdn.playbuzz.com/cdn/375008dd-b679-4825-8046-ce70cc960250/ce620679-2e9c-44ff-81a1-02cbcee1fc3b.png | Backspace will restart the song from the beginning |



**CONTROLS**

**The Following table displays the keys used to control playback**