JETSON'S

Jetson's application basically plays the entire Jetson's theme song as a whole. Our available data is basically 23 separate raw wave files containing small sections of the song. Our multithreaded application basically plays all 23 wave one by one without any gaps, making it appear as a whole song.

My application has 28 threads in total. 23 of these threads are Wave threads and the rest areMain, File, Coordinator, Kill and Playback threads. All these threads are made in the main file. The wave threads are created in the main file but are launched in the Playback thread so as to only launch the thread when it is needed.

The file thread basically does the work in the File System. It is used to load the files one by one. The file system uses win32Api file functions such as Open, Read, Write, Seek and Tell. The file thread keeps loading the files one by one and gets the buffer data and size of the file. The buffer data is then sent to Coordinator Buffers namely A/B or Front/Back. The file system has a conditional variable which notifies to the wait in the Buffer Management system in the Coordinator Thread class.

The Coordinator thread basically obtains the data from the File system and fills the wave buffer in the wave thread. The Buffer Management System has 2 Buffers. It is also capable of swapping those 2 Buffers.

The Playback Thread launches the wave threads. The Playback thread gets the buffer from the A or B buffers and inserts them with the Wave thread in a circular queue. The Queue then starts popping the buffers one by one and starts executing each wave buffer one by one by writing each buffer to the wave callback. The wave callback then starts playing each wave one by one. The Sample rate is 22050, 16 bits per sample with stereo sound channel.

JETSON'S

Everything is done synchronously thanks to multithreading. Every thread is joined at the destruction of each thread class. Every thread also has a kill function so that a key press basically kills all the threads except the main thread.

Communication

The file thread communicates with the Buffer management with the help of condition variables. The condition variable notifies when a file is loaded and is then loaded to A/B buffers which wait until it is notified by the Condition Variable in the File System.

All the threads have a promise to kill and since all the threads have one common promise they all have a shared future to kill. The kill is executed by using getch() which is obtained from the conio header file. Once any key is pressed in the keyboard all the threads are then killed after 10 seconds because of the sleep function so as to avoid any race conditions and not crash.

The main thread is not killed so as to avoid any system crashes so there is a separate promise for the main thread.

Mutex and Lock Guards were used frequently namely when writing the buffer to the wave callback and killing threads.

Mutexes and lock guards were very useful for protections and helped a lot even in debugging.

Issues

One of the issues I had with the Jetson's project was the Playback. Creating the Playback thread and later synchronously pulling the data from the buffer and creating a wave queue and eventually write the data Buffer to the Wave Callback took some time to figure out. I realized that I did not know well enough how the wave threads worked so I had to learn and experiment the Write and Kick function in the Wave Thread.

JETSON'S

Another problem I faced was after every wave there was a non-uniform gap. What confused me was that there was no gap between the first wave and the second wave. I later found out that the buffer was uniform but the waves were not the same size. So I eventually used the buffer size as a parameter when writing the buffer to the wave.

I was able to learn a lot of stuff about multithreading . Using Promise, Shared futures and future was a valuable experience. I also learnt about how Callbacks work and also how to use Circular Queues effectively. Working with file systems and synchronous loading, pulling and pushing data to the buffers. Debugging in threads was a bit of a challenge but it was a very good experience. It was really fun to see a multi-threaded process where there was loading files, pulling and pushing buffers, creating queues and also playing a fun song all working together at the same time avoiding any edge conditions and playing seamlessly. Finally I was able to learn how multithreading works and how to use it properly.