ABSTRACT (DRAFT)
The Document describes the detailed design of mp3 encoder.

Tapeswar Puhan (Senior C++ developer)

Mp3Encoder

1. INTRODUCTION

1.1 Purpose

The application provides user interface to decode Wave to mp3 decoding.
The Encoder only supports for LINUX.

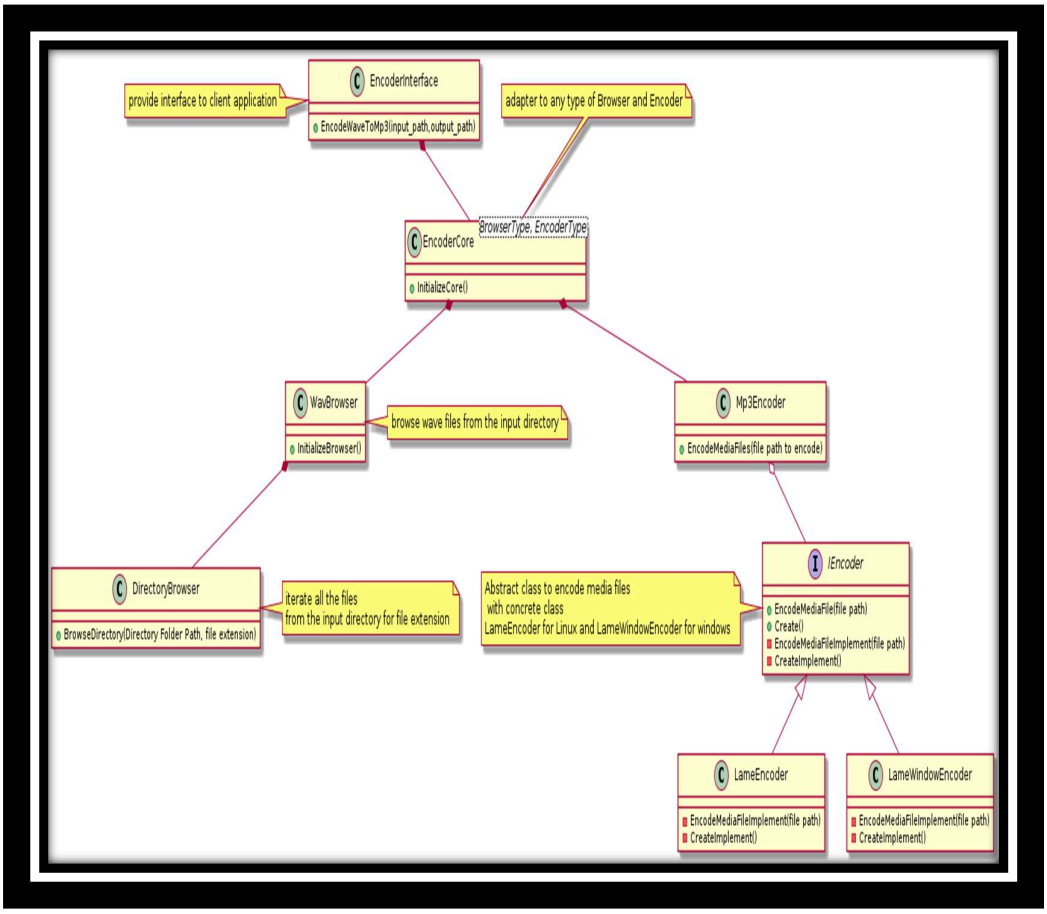
1.2 Scope

The design of application gives scope to adopt many other type of encoder for example Wave to Mp4, Wave to Avi in future.
A new encoder can be implemented future to adopt existing core application.

Note: Window Encoder is not implemented for this version of application,
window encoder can be implemented and adopted to existing application in future.

2. SYSTEM ARCHITECTURE

2.1 Class Diagram



- **EncoderInterface :**

Provides Interface to user to encode mp3 files from wave format.

Interface: EncodeWaveToMp3(const DirectoryName & input_directory_path, const DirectoryName& output_directory_path)

Param 1: input Directory path.

Param 2: output Directory path

➢ EncoderCore:

Template class can adopt any type of parser and encoder. The design of this class will help client application to add new types of browser and encoder.
In the existing application, we have WavBrowser and Mp3Encoder.
WavBrowser will browse wave files available in the folder.
Mp3Encoder encode all the wave files to mp3 files.

Interface :-
InitializeCore()

Config information: -

MAX_NUMBER_TASK - Application can set how many threads needs to run parallel, now value is 10.

MAX_BUFFER_SIZE: - Set the buffer size limit which holds the input file path. This will help application to prevent overflow if the browser task is processing rate is more than encoder task.

THRESHOLD_BUFFER_SIZE: - Notify the browser thread to start again if it blocked due to maximum buffer limit.

➢ WavBrowser:

Browse wave files from the input directory path.
Interface:-

- InitializeBrowser();

Initialize the wave browser initialize the browser thread.

Initialize DirectoryBrowser to read directory to get wave files from the call back function.

- RegisterForMediaFileUpdates(UpdateMediaFilesHandler&& media_files_update_handler)

UpdateMediaFilesHandler :- call back function to get list of wave files in the input folder.

Config information: -

MAX_FILE_LIST: - It suggests how many number of file needs to updated at a time. If value is 1, the update happens one by one.

Currently, it updates 5 wave files at a time.

➢ DirectoryBrowser

Interface:

- BrowseDirectory(const DirectoryName& input_directory, const std::string& file_extension)
  Browse the directory with provided extension name.

- RegisterTOUpdateFileName (UpdateFileNameHandler&& file_name_handler)
  Provide user to register its callback function to get file path updates.

➢ IEncoder

Abstract class provides interface to encode the media files. A new concrete class can be derived to encode new type of encoding.

Interface:

EncodeMediaFile (MediaFileName&& file_name)

Command to encode media files.

RegisterUpdateInformation (UpdateInformation&& update_info)

Get Information about the encoding status.

➢ LameEncoder

Use Lame Library to encode mp3 files for Linux platform which is concrete class of IEncoder.

➢ LameWindowEncoder(TBD)

Use window encoder Library to encode mp3 files for Window platform which is concrete class of IEncoder.
Note: - The implementation is not available with this version of application.

➢ WavHeader :

Extract the Wave header from input wave file.

➢ Mp3Encoder

Encode mp3 files by using IEncoder.

3. Sequence DESIGN

Encoder Interface (EncodeWaveToMp3) -> EncoderCore< WavBrowser, Mp3Encoder>

EncoderCore get wave file path from WavBrowser.

Using Mp3Encoder encode wave files to mp3 files.

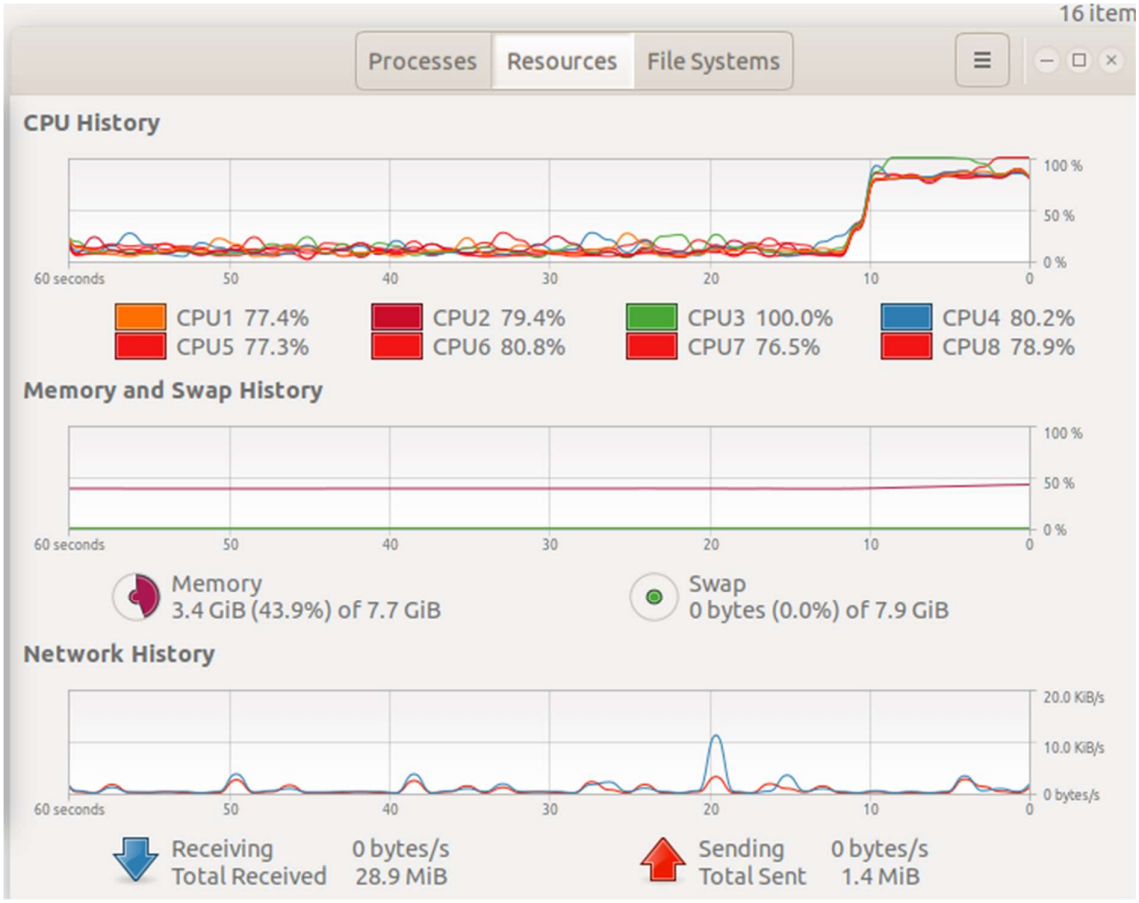
4. Thread DESIGN

List of threads running.

1. Browser thread:- To read wave files from the input folder path
2. Thread Pool:- creates number of worker threads to process encoding parallelly
3. A polling thread to read the input buffer and create a worker thread which will be a part of thread pool.

Thread pool :-

Provides interface AddTask which stores the task in thread safe queue.
Each task should be executed by its own encoder to avoid race condition (For parallel execution and maximum utilize of CPU no lock mechanism is used for worker threads to decode)



5. BUILD And Execute

Step 1: Go to folder wave_to_mp3_encoder
Step 2: cmake
Step3: make
Executable name mp3encoder

$ mp3encoder “path”
To browse the wave files from the path
$ mp3encoder
If no path is provided, the application considers the current path

6. Test

Application was tested taking into account the functional behaviour and the application passed all tests successfully. Below are few high-level functional test cases.

Test Case Name	Test Case Description	Expected	Actual
TC_ENCODE_WAV_MP3_001	Check the behaviour of the application if no wave files are available in the input path.	The application should run normally without any abnormal behaviour.	The application ran normally without any unexpected behaviour.
TC_ENCODE_WAV_MP3_002	Check the behaviour of the application if only one wave file name **file.wav** is available the input path $ mp3encoder ./test	Corresponding mp3 should be generated (file.mp3). Playback should be same as file.wav	Corresponding mp3 was generated (file.mp3). Playback was same as file.wav
TC_ENCODE_WAV_MP3_003	Check the behaviour of the application if multiple wave files are available the input path $ mp3encoder ./test	Corresponding mp3 should be generated.	Corresponding mp3 was generated.
TC_ENCODE_WAV_MP3_004	Check the behaviour of the application if multiple Wave files are available the input path and also wave files are available in subfolders. $ mp3encoder ./test	Corresponding mp3 file should be generated recursively.	Corresponding mp3 file was generated recursively.
TC_ENCODE_WAV_MP3_005	Check the behaviour of the application **(stress testing)** by keeping 10000 wave files in the input folder.	Corresponding mp3 should be generated with no abnormal behaviour. CPU should utilize all its cores.	Corresponding mp3 was generated. CPU should utilize all its cores.