Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Implementación de una aplicación en lenguaje C para la obtención y visualización de señales de sonido en entorno GNU Linux

Autor: Miguel Ciurana Tomás
Tutor: José Crespo Del Arco

Madrid, Junio - 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Implementación de una aplicación en lenguaje C para la obtención y visualización de señales de sonido en entorno GNU Linux

Junio - 2024

Autor: Miguel Ciurana Tomás
Tutor: José Crespo Del Arco
Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Índice general

1. Introducción **3**
1.1. Contexto del proyecto 3
1.2. Objetivos 3
1.3. Estructura de la memoria 4

2. Estado del Arte **5**
2.1. GNU/Linux 5
2.2. Lenguaje C 5
2.3. Software relacionado 6
2.4. Formatos de audio 7

3. Entorno y herramientas empleadas **8**
3.1. Distribución utilizada 8
3.2. Librerías empleadas 8

4. Desarrollo **10**
4.1. Formato de audio utilizado 10
4.1.1. Formato WAVE 10
4.2. Frecuencias de muestreo 13
4.3. Canales en el audio 14
4.4. Captura de audio en sistemas GNU Linux 15
4.5. Reproducción de audio en sistemas GNU Linux 16
4.6. Transformada rápida de Fourier 17
4.7. Conversión de formato de archivo de audio 19
4.8. Limitaciones y problemas encontrados 20

5. Impacto del trabajo **21**

6. Resultados y conclusiones **22**
6.1. Resultados 22
6.2. Conclusiones personales 23
6.3. Trabajo futuro 23

Bibliografía **24**

A. Anexo **26**
A.1. Manual de usuario 26
A.2. Informe de originalidad 29

Índice de Figuras

4.1. Formato de archivo .wav 11
4.2. Ejemplo de segmento de archivo .wav 12
4.3. Onda sinusoidal muestreada 13
4.4. Diagrama secuencial del proceso de captura 15
4.5. Diagrama secuencial del proceso de reproducción 16
4.6. Visualización de las frecuencias en la pantalla de reproducción 19

A.1. Estructura carpetas 26
A.2. Pantalla inicial aplicación 27
A.3. Pantalla de reproducción 28

Resumen

En este trabajo se busca realizar un estudio de diversos conceptos relacionados con archivos y señales de audio, así como de los formatos y dispositivos asociados.

En la actualidad, las señales de audio son una parte fundamental de nuestra vida, presentes en la música que escuchamos o en las llamadas telefónicas que realizamos. Procesar y manipular estas señales es crucial en diversos campos de la tecnología actual.

Debido a esta gran importancia, existen múltiples formatos de audio distintos. Este trabajo se va a centrar en el formato de audio WAVE y cómo éste se captura y reproduce en sistemas GNU/Linux.

Además, se desarrollará una implementación en lenguaje C de una aplicación que permita tanto la captura como la reproducción de señales de audio. Todo esto en un entorno GNU/Linux. Así mismo, la aplicación permitirá la conversión de las señales de audio entre distintos formatos y poder ver la información de los audios capturados.

Por último, se elaborará una documentación asociada con la que el usuario pueda comprender la aplicación y como ha de usarse.

Con este proyecto, se espera desarrollar una herramienta útil y educativa con la que poder comprender como los sistemas GNU/Linux manejan las señales de audio digitales.

Abstract

This work aims to conduct a detailed study of various concepts related to audio files and signals, as well as associated formats and devices.

In our daily lives, audio signals play a very important role, such as the music we listen to or the phone calls we make. Processing and manipulating these signals is crucial in various fields of modern technology.

Due to this great importance, there are multiple different audio formats. This work will focus on the WAVE audio format and how it is captured and reproduced in GNU/-Linux systems.

In addition, an implementation in C language will be developed for an application that allows both the capture and playback of audio signals. All this will be done in a GNU/Linux environment. Likewise, the application will enable the conversion of audio signals between different formats and the ability to view the information of the captured audio.

Finally, associated documentation will be created to help users understand the application and how to use it.

This project aims to develop a useful and educational tool to understand how GNU/-Linux systems handle digital audio signals.

Capítulo 1

Introducción

1.1. Contexto del proyecto

Actualmente, las señales de audio son una parte clave en nuestro día a día, desde las llamadas de teléfono o los audios que mandamos por aplicaciones de mensajería hasta estudios médicos [1]. El procesado y manipulación de las señales de audio de manera eficiente es muy importante en muchos campos de la tecnología actual.

En este trabajo, se propone la creación de una aplicación para sistemas GNU/Linux que permita tanto la captura como la visualización de señales de audio. Esto se realizará utilizando librerías nativas de C y de terceros. Esta aplicación dispondrá de una interfaz sencilla e intuitiva con el fin de ser accesible para una gran variedad de usuarios. No obstante, también dejando la opción de una utilización más cómoda desde la línea de comandos, de tal manera se permitirá también a los usuarios avanzados poder realizar esas mismas tareas.

1.2. Objetivos

El objetivo de este trabajo es el desarrollo de una aplicación en lenguaje C centrada en la captura y visualización de señales de audio para sistemas Linux. Al inicio del trabajo se plantearon los siguientes objetivos:

- Estudio de conceptos referentes a archivos y señales de audio, formatos y dispositivos asociados.
- Revisión de algunas iniciativas relacionadas realizadas en lenguaje C.
- Implementación de funcionalidad para capturar y visualizar señales de audio.
- Realización de ejemplos descriptivos del código y documentación asociada.
- Desarrollo de aplicación con interfaz de línea de comandos para poder ser ejecutada desde la consola.

Durante el desarrollo del trabajo surgieron, además de los ya establecidos, nuevos objetivos:

- Integración de función para visualizar señales de audio mediante la transformada rápida de Fourier.
- Implementación de función de conversión de formato de archivos de audio.
- Realización de implementación para visualizar la información de un archivo de audio.

1.3. Estructura de la memoria

En el segundo capítulo se trata el estado del arte, donde se examinan las tecnologías existentes como aplicaciones o librerías. También se trata el lenguaje de programación con el que se desarrollará la aplicación y se realiza una introducción a los tipos de formatos de audio.

En el tercer capítulo se abordan tanto la distribución de GNU/Linux con la que se ha realizado el trabajo. Como las librerías que se han empleado para el desarrollo de la aplicación.

En el cuarto capítulo se trata el desarrollo del trabajo, se tratan diversos conceptos como el formato de audio que se ha utilizado durante la realización de este trabajo. Cómo los sistemas Linux capturan y reproducen audio. Y diversos conceptos relacionados con el audio, como la frecuencia de muestreo, los canales en el audio o la transformada rápida de Fourier.

En el quinto capítulo se considera el impacto que puede llegar a tener este trabajo. Además se explora cómo este trabajo contribuye al cumplimiento de los Objetivos de Desarrollo Sostenible de la Agenda 2030.

En el sexto capítulo se detallan los resultados obtenidos en la realización de este trabajo y se tratan unas conclusiones personales sobre el desarrollo de este TFG. Además se detalla el trabajo a futuro que se realizará sobre la aplicación.

En el Anexo se encuentra un manual de usuario para entender el funcionamiento de la aplicación.

Capítulo 2

Estado del Arte

2.1. GNU/Linux

Los sistemas GNU/Linux son una familia de sistemas operativos de código abierto basados en el núcleo Linux. Fue creado en el año 1991 por Linus Torvalds en la Universidad de Helsinki [2]. Muchos sistemas utilizan GNU/Linux, desde dispositivos integrados como televisores hasta teléfonos móviles con Android, además de contar con múltiples distribuciones para computadoras. GNU además cuenta con una gran comunidad que desarrolla o mejora las distribuciones más usadas.

Existen múltiples distribuciones GNU/Linux como Red Hat, Debian, Fedora o Ubuntu. Todos estos sistemas destacan por diversas razones [3]. Estos sistemas son de código abierto, lo que significa que cualquier persona podría modificar el sistema operativo. Esto permite a los sistemas GNU/Linux ser altamente personalizables. Además, estos sistemas son muy estables. Debido a esto, muchas supercomputadoras utilizan Linux, ya que así pueden funcionar durante largos periodos de tiempo [4].

Por otro lado, se encuentran los sistemas UNIX [5]. UNIX es un sistema operativo que fue desarrollado en el año 1969 en los Laboratorios Bell de AT&T. Del sistema UNIX surgieron diversos sistemas operativos, como macOS, el sistema operativo que utiliza Apple para sus ordenadores. Sin embargo, hay que mencionar que Linux no es un sistema UNIX; este solo se inspiró en él, haciéndolo así parecido a otros sistemas UNIX pero no siendo parte de ellos.

2.2. Lenguaje C

C es un lenguaje de programación desarrollado por Dennis Ritchie entre los años 1969 y 1972. Desde el principio, este lenguaje ha estado muy ligado a los sistemas Linux, ya que fue originalmente diseñado con el fin del desarrollo de este núcleo [6].

La razón por la que se ha elegido C como el lenguaje con el que se va a realizar el desarrollo y no con otros lenguajes populares [7] como Python o Java se debe principalmente a una. El nivel de programación en C.

Actualmente, C es considerado un lenguaje de bajo nivel debido a que, por ejemplo, el propio usuario es el que se ha de manejar la asignación y liberación de memoria para los objetos.

Sin embargo, en este caso, el bajo nivel de C como lenguaje es lo que resulta útil. Esto se debe a que en C, como se tiene un mayor control del hardware, se puede tener un control más preciso sobre la reproducción, captura y procesamiento del audio. Además, este idioma de programación destaca por su rendimiento y eficiencia, lo que lo hace adecuado para este trabajo, ya que las señales de audio se procesan en tiempo real. Aún con todo, este mayor control también supone un problema.

Esto se atribuye a que utilizar C también conlleva sus riesgos. Ya que para hacer uso de C correctamente, uno tiene que entender qué está haciendo o puede acabar teniendo problemas como desbordamientos de búfer o fugas de memoria. Esto es visible por comunicados dados por la Casa Blanca [8].

2.3. Software relacionado

Actualmente, se presentan diversos software centrados en múltiples plataformas, como Windows o macOS. Un ejemplo es Adobe Audition [9], un software diseñado para sistemas Windows y macOS. Aunque está enfocado en la edición de señales de audio, también proporciona funcionalidades de captura y reproducción de audio en tiempo real. Otro programa disponible para ambos sistemas operativos es Pro Tools [10], que aunque su enfoque principal esté en la grabación, edición y mezcla de audio, también permite la visualización y captura de señales de audio. Dejando visualizar la forma de la onda de las señales y grabar audio en tiempo real desde varias fuentes.

En sistemas GNU Linux, se presentan programas como Audacity [11], una aplicación de software libre y de código abierto centrada en la grabación y edición de audio. A su vez, PulseAudio [12], que aunque es un servidor de sonido gestiona el audio de sistemas Linux, también puede ser utilizado desde la línea de comandos e incluso cuenta con una interfaz gráfica. Por último, otro programa reseñable es FFmpeg [13], un software multiplataforma centrado en la grabación, conversión y reproducción de archivos de audio en distintos formatos.

Para lenguaje C, se presentan diversas librerías centradas en lo referente al audio, una de ellas siendo miniaudio [14]. Una biblioteca escrita en C en un solo archivo y además multiplataforma. Siendo ésta una librería centrada principalmente en la captura y reproducción de archivos de audio. Otra librería reseñable es dr_wav [15], una biblioteca en un solo archivo que permite leer y escribir archivos .wav en C.

Un último software a destacar es Musializer [16], un programa centrado en la visualización de archivos de audio mediante la transformada rápida de Fourier y que hace uso de la biblioteca de miniaudio y de raylib para el desarrollo de su interfaz. Dos bibliotecas que han sido también utilizadas en el desarrollo de este proyecto.

Finalmente, para el desarrollo de interfaces en lenguaje C, se presentan diversas librerías. Las que más han afectado al desarrollo de este trabajo han sido GTK [17], una librería multiplataforma y con mucha documentación. La otra librería a destacar es raylib [18], otra biblioteca también multiplataforma enfocada en la programación de videojuegos. Ambas librerías han afectado al desarrollo debido a que inicialmente, la aplicación se desarrolló utilizando GTK. Sin embargo, se acabó optando por raylib.

2.4. Formatos de audio

Existen dos tipos de formatos de audio. Con pérdidas y sin pérdidas. Los archivos de audio con pérdidas utilizan diversos métodos para realizar la compresión. En uno de ellos, comprimen el audio eliminando la información que se considera inaudible para el oído humano. Este formato hace que se ahorre espacio de almacenamiento, ya que los archivos de audio sin pérdidas pueden llegar a ocupar mucho espacio. Cabe mencionar que una repetida compresión del mismo archivo puede llevar al degradado del audio. Ya que cada vez que se descomprime y se vuelve a comprimir un archivo se elimina más información y se introduce más distorsión [19]. Varios ejemplos de este tipo de formato de audio podrían ser MP3, Opus o AAC [20].

Por otro lado, en los archivos de audio sin pérdidas no se pierde nada durante la compresión. Estos archivos pueden tanto estar descomprimidos (WAV, AIFF) como comprimidos (FLAC, ALAC). Se puede observar, que estos archivos pueden llegar a ocupar mucho espacio y una canción de cuatro minutos en formato WAVE puede hasta llegar a ocupar 50MB.

Capítulo 3

Entorno y herramientas empleadas

3.1. Distribución utilizada

Dentro de la familia GNU Linux, existen diversas distribuciones como Debian, Fedora, Red Hat o Ubuntu con las que se podría haber desarrollado el proyecto. Sin embargo, se decidió emplear Ubuntu como sistema operativo elegido. Esto se debe a diversas razones.

La primera es que durante la realización de la carrera de ingeniería informática se ha familiarizado a los alumnos con Ubuntu en varias asignaturas. Además de eso, Ubuntu cuenta con una interfaz de usuario amigable y es compatible con una gran cantidad de aplicaciones. Por último, Ubuntu cuenta con una gran comunidad a la que se puede recurrir en caso de tener problemas con el sistema. Cabe destacar que debido a estas razones Ubuntu se ha convertido en una de las distribuciones de GNU/Linux más utilizadas [21].

3.2. Librerías empleadas

Para el desarrollo de la aplicación se han empleado diversas librerías externas de C. Una de ellas es miniaudio [14], una biblioteca enfocada en la reproducción y captura de audio. Hay múltiples razones por las que se ha elegido miniaudio en vez de otras bibliotecas como PortAudio [22], OpenAL [23] o dr_wav [15] .

Principalmente, esto se debe a la sencillez de miniaudio. Miniaudio está escrita en un único archivo por lo que su instalación es muy sencilla. Además de esto, al ser un único archivo, este no va a ocupar mucho espacio en memoria, compensando así el espacio que ocuparán los archivos .wav. También, miniaudio es muy fácil de integrar en el código y tiene un extenso catálogo de ejemplos [24] con los que el usuario puede entender fácilmente la librería. Por último, al centrarse en la reproducción y captura de sonido, esta carece de las complejidades que las demás bibliotecas tienen al estar centradas en más funciones que solo esas dos.

Aún con todo el uso de miniaudio también conlleva ciertas desventajas. La primera es que si se quiere obtener la información del búfer del audio para, por ejemplo, poder realizar la transformada rápida de Fourier, se ha de usar un objeto *decoder*. Este objeto funciona bien con archivos .wav, pero con otras extensiones como .mp3 o .flac con las que también debería poder, tiene problemas a la hora de extraer esa información. Aún con todo, miniaudio puede reproducir correctamente el audio, sólo que tiene problemas para extraer las muestras de éste.

Además, como se ha mencionado anteriormente el codificador de miniaudio sólo permite la codificación del audio generado a .wav. Por lo que al usar miniaudio en vez de otras librerías uno se cierra a utilizar más formatos que el WAVE.

Para la implementación de interfaz de usuario se ha usado la librería de raylib [18], una biblioteca de C diseñada para el desarrollo de aplicaciones gráficas y videojuegos. Cuenta con diversos módulos como raudio, un modulo de audio el cual implementa raylib. O rshapes un módulo enfocado en el renderizado de formas 2D básicas. Se ha elegido raylib frente a otras librerías como GTK [17] debido a diversas razones.

Esta elección se debe fundamentalmente a la facilidad con la que uno puede aprender a utilizar raylib. Ambas librerías pueden implementar interfaces de usuario, sin embargo raylib cuenta con una gran cantidad de ejemplos con los cuales se puede aprender a utilizar la librería. También, la librería tiene un gran enfoque en ser fácil de aprender y utilizar. Además de raylib, también se ha hecho uso de raygui [25]. Una librería complementaria a raylib, la cual está enfocada en la creación de interfaces gráficas de usuario.

Capítulo 4

Desarrollo

4.1. Formato de audio utilizado

Como se ha visto anteriormente, existe una variedad bastante amplia de formatos de audio con los que se puede trabajar. Sin embargo, se ha decidido trabajar únicamente con los archivos .wav, esto se debe a dos razones:

- En comparación con otros archivos de audio, los archivos .wav son especialmente sencillos de comprender debido a su estructura.
- El codificador de miniaudio solo permite la codificación del audio capturado a formato .wav.

Aún con todo, trabajar con estos archivos supone también desventajas. Principalmente, el espacio en memoria que pueden llegar a ocupar, ya que como se ha mencionado previamente, este formato tiene la capacidad de ocupar bastante memoria.

4.1.1. Formato WAVE

Los archivos .wav se componen de dos secciones: una cabecera donde se detalla toda la información del archivo y una sección donde están representadas las muestras del audio.

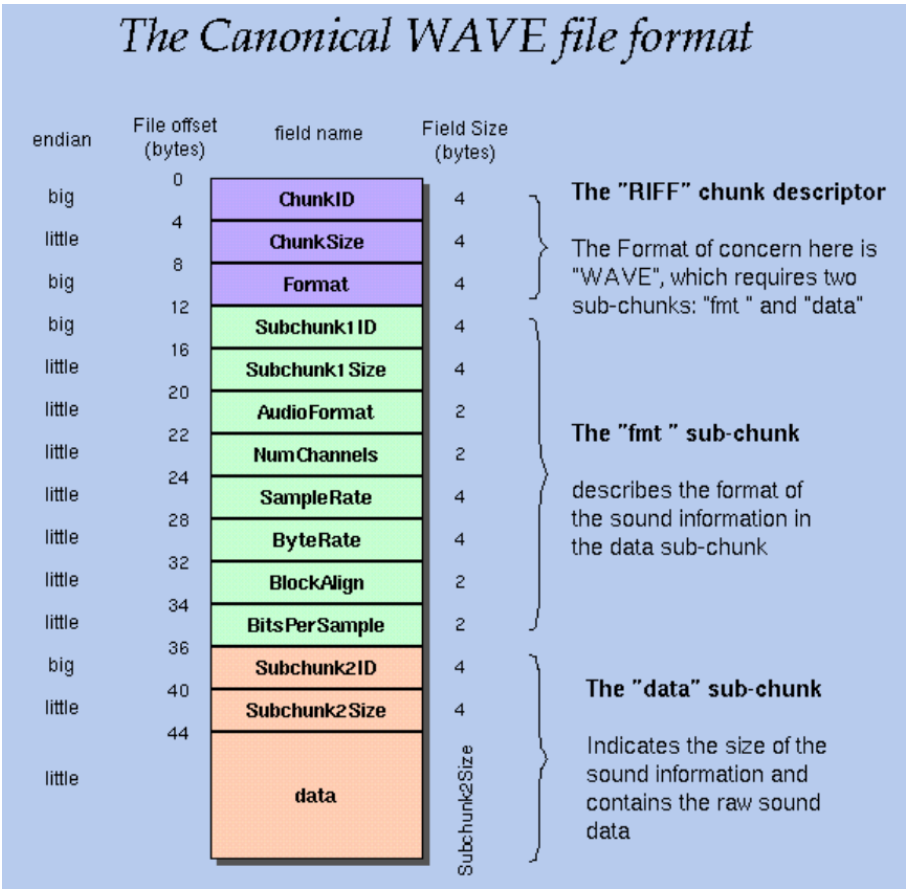

Figura 4.1: Formato de archivo .wav

Tanto la figura 4.1 como la figura 4.2 han sido obtenidas del siguiente enlace [26]. Todo el contenido del archivo .wav está escrito en datos binarios, que pueden ser representados en hexadecimal. Como se puede ver en la figura 4.1, hay valores que están escritos en big endian y otros en little endian, dos formas distintas de ordenar los bytes dentro de un dato. En los valores big endian, los bytes más significativos se almacenan en las direcciones de memoria bajas, mientras que en little endian es al revés, los bytes más significativos se almacenan en las direcciones de memoria altas. Debido a esto, hay que tenerlo en cuenta una vez se procese el archivo.

La cabecera se divide en las siguientes partes:

- **Identificador del trozo** (Chunk ID): Contiene los caracteres “RIFF”. Marca que el archivo es del tipo RIFF [27].
- **Tamaño del trozo** (Chunk Size): Indica el tamaño del archivo en bytes.
- **Formato** (Format): Indica el formato del archivo, para el caso de los archivos .wav contendrá "WAVE".
- **Identificador del subtrozo 1** (Subchunk1 ID): Contiene los caracteres "fmt", indica el comienzo del trozo de formato.

- **Tamaño del subtrozo 1** (Subchunk1 Size): Indica el tamaño en bytes del subtrozo 1, contiene "16".
- **Formato de audio** (Audio Format): Indica el método sin pérdida utilizado para almacenar audio.
- **Número de canales** (Num Channels): Indica el número de canales del audio, si el archivo solo tiene un canal se denomina "mono", si tiene dos “estéreo”.
- **Frecuencia de muestreo** (Sample Rate): Indica el número de muestras que tiene el audio por segundo, se mide en hercios. Esto quiere decir que si, por ejemplo, un audio tiene una frecuencia de 44100Hz se tomarán 44100 muestras de audio por segundo. Una muestra representa una instantánea de la amplitud de la señal del audio en un momento específico.
- **Tasa de bytes** (Byte rate): Indica la cantidad de bytes que se utilizan por segundo para almacenar los datos de un audio.
- **Alineación de bloques** (Block align): Indica el número de bytes de una muestra.
- **Bits por muestra** (Bits per sample): Indica el número de bits por muestra. A mayor número de bits, el rango de la muestra del audio es mayor y la calidad del audio es mejor. El estándar son dieciséis bits.
- **Identificador del subtrozo 2** (Subchunk1 ID): Contiene los caracteres "data", indica el comienzo del trozo de los datos.
- **Tamaño del subtrozo 2** (Subchunk1 Size): Indica el tamaño en bytes del subtrozo 2.
- **Datos** (data): Sección que contiene todas las amplitudes de la señal del audio. Generalmente tienen un rango de -32768 a 32767, este rango depende de los bits por muestra.

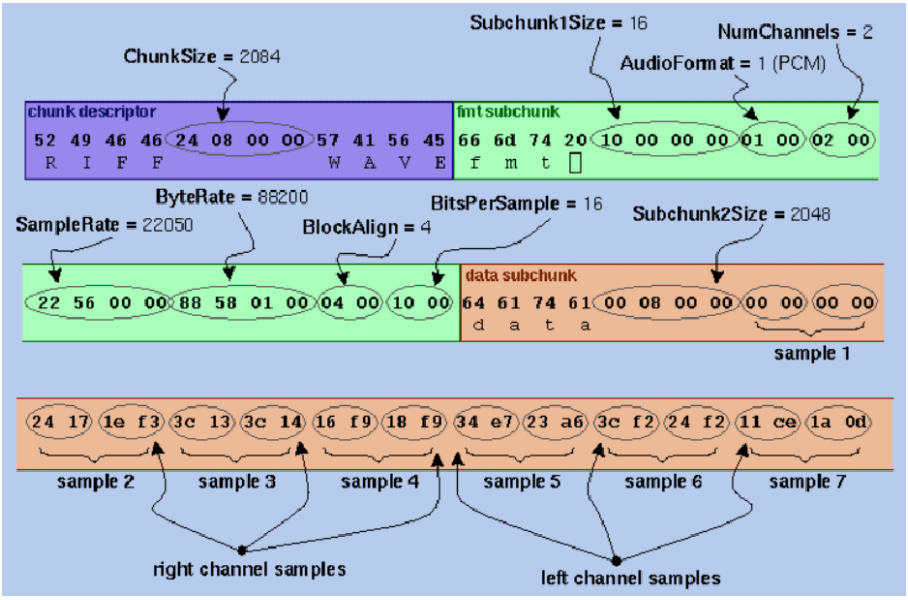

Figura 4.2: Ejemplo de segmento de archivo .wav

En la figura 4.2 se puede ver un ejemplo de un segmento al inicio de un archivo .wav. Como se puede observar, todos los datos del archivo están representados en hexadecimal. Para poder tratar todos estos datos, hay que transformarlos de su formato binario, también hay que tener en cuenta qué datos están en big endian y cuales en little endian.

4.2. Frecuencias de muestreo

Dos de las frecuencias de muestreo más comunes son 44100Hz y 48000Hz. La primera es usada en los audios grabados en CDs y ambas son utilizadas en diversas situaciones como en las tarjetas de sonido de los ordenadores o en la televisión.

Como se ha mencionado anteriormente, la frecuencia de muestreo es el número de veces que se captura una muestra por segundo. Cada muestra representa la amplitud de la señal del audio en el momento en el que se ha realizado la captura.

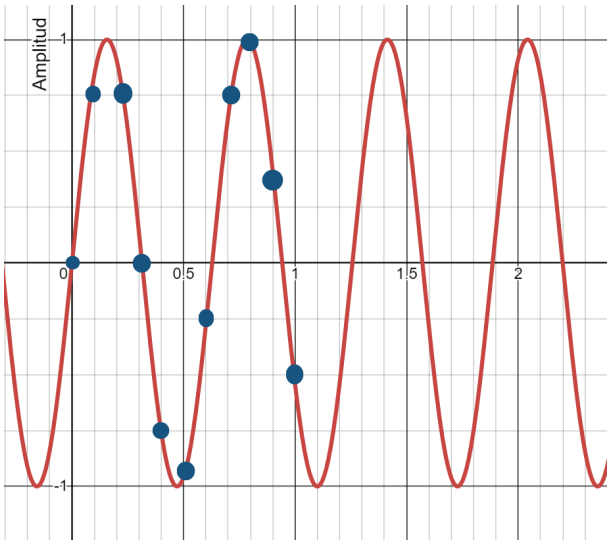

Figura 4.3: Onda sinusoidal muestreada

Dada la figura 4.3, en la que el eje Y representa la amplitud de la onda y el eje X representa el tiempo, se puede observar que hay diez puntos azules desde el segundo cero hasta el segundo uno. Estos puntos representan las muestras que se tomarían si la onda de sonido fuese capturada con una frecuencia de muestreo de 10 hercios.

Tras saber qué es la frecuencia de muestreo, surge la cuestión de por qué 44100Hz y 48000Hz son los más comunes. Aquí es donde entra el teorema de Nyquist-Shannon, el cual establece que la frecuencia de muestreo debe ser al menos el doble de la frecuencia máxima de la señal para que esta pueda ser reconstruida sin que se produzca pérdida de información.

Si la señal se muestrea a una frecuencia menor que el doble de la frecuencia máxima de la señal, se producirá un fenómeno conocido como aliasing. Cuando se produce aliasing, los componentes de alta frecuencia de la señal análoga no serán interpretados correctamente, por lo que la señal digital no representará fielmente la señal analógica.

El rango de hercios que un ser humano puede llegar a escuchar es de 20Hz a 20.000Hz [28]. Debido a que la frecuencia máxima son 20.000Hz, según el teorema de Nyquist-Shannon, para poder reconstruir una señal que el ser humano pueda oír, la frecuencia de muestreo debería ser al menos el doble de la frecuencia máxima. Por lo tanto: $f = 2 \cdot 20.000Hz = 40.000Hz$. Debido a esto se utilizan diversas frecuencias de muestreo como 44100Hz o 48000Hz. La elección de qué frecuencia usar se debe a factores como la calidad del audio deseada, la potencia del equipo o la compatibilidad con otros dispositivos.

4.3. Canales en el audio

Cuando se graba o se reproduce audio, los canales de audio son un componente esencial. Un canal de audio representa un punto desde el cual el audio se recibe o envía. Por ejemplo, un audio con dos canales (estéreo) puede ser escuchado con unos auriculares y a cada oreja llegará una señal de audio distinta, una por cada canal.

Existen múltiples tipos de formato de canal, sin embargo los más comunes son los siguientes:

- **Monoaural**: El sonido se captura y se reproduce por un solo canal. Actualmente no se utiliza excepto para radios AM y algunas redes telefónicas.
- **Estereofónico**: El sonido es capturado y reproducido por dos canales, los canales se denominan izquierdo y derecho. Es el formato de audio más común.
- **Envolvente**: Múltiples canales de audio son utilizados (canales 5.1, por ejemplo) con el fin de generar un sonido envolvente. Es comúnmente utilizado en películas o videojuegos para crear una experiencia inmersiva.

Los canales 5.1 consisten de tres canales frontales: izquierdo, derecho y central. Dos canales traseros: izquierda y derecha, y un canal de bajos con el cual se reproducen sonidos graves y de baja frecuencia. Existen más tipos de canales envolventes; todos siguen distribuciones similares al canal previamente explicado.

4.4. Captura de audio en sistemas GNU Linux

En la siguiente figura 4.4 se representa el proceso de captura de audio de la aplicación mediante un diagrama secuencial. La flecha roja indica que ese proceso es continuo hasta que se produzca el aviso de fin de captura.

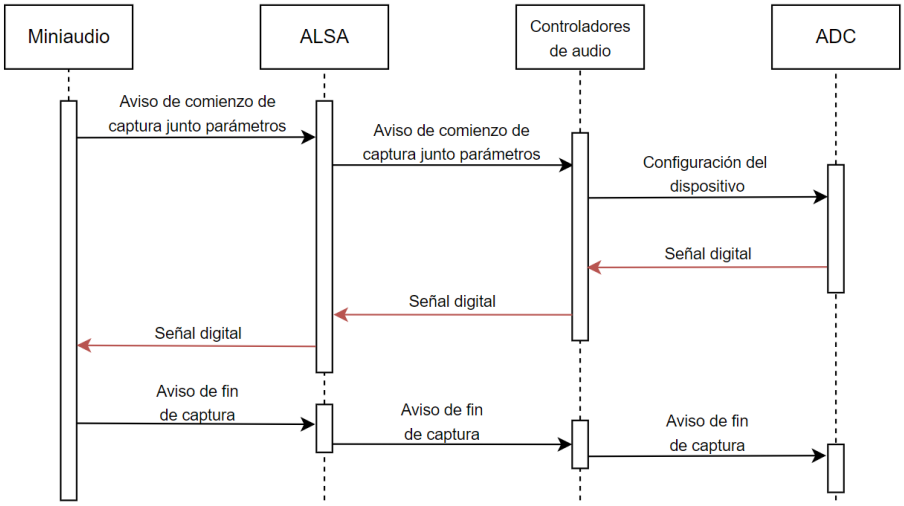

Figura 4.4: Diagrama secuencial del proceso de captura

El proceso de captura de audio empieza desde el micrófono. Aunque existen diversas tecnologías para capturar el sonido analógico todas se basan en el mismo principio. El transductor, una pieza del micrófono, convierte las ondas del sonido en señales eléctricas mediante la vibración que este genera. Antes de que la señal eléctrica se pase por el ADC, estas pasan por el filtro de paso bajo. Este filtro se encarga de descartar las frecuencias que estén por encima de la mitad de la frecuencia de muestreo con la que se está realizando la captura. Por ejemplo, si la frecuencia de muestreo es de 48kHz, el filtro de paso bajo eliminará las frecuencias por encima de los 24kHz. Esto se realiza para evitar la aparición de aliasing en el audio capturado. Posteriormente, la señal eléctrica es pasada a un conversor de señal analógica a digital (ADC). Este dispositivo se encarga de medir la amplitud y frecuencia de la señal para convertir estas medidas a números binarios, representando así la señal en binario. El ADC obtiene las muestras de la señal en intervalos regulares, estas vienen definidas por la frecuencia de muestreo que se le solicite al conversor.

Una vez obtenida esta información en formato binario, los controladores de audio del dispositivo reciben las muestras digitales. Estos controladores son específicos del hardware y son los responsables de la gestión de la interacción con el hardware del audio. Posteriormente, estos controladores facilitan las muestras a ALSA (Advanced Linux Sound Architecture) [29]. ALSA es la arquitectura de sonido de sistemas Linux, es un software de nivel muy bajo y se encarga de la conexión con los controladores de audio en GNU/Linux. Dicha arquitectura, al recibir las muestras digitales, las pone a disposición de las aplicaciones que las necesiten.

En el contexto de este trabajo, para poder capturar sonido a través de miniaudio, se seguiría un orden similar sólo que con un paso más. Al inicializar el codificador en miniaudio, a este se le especifica la frecuencia, los canales y el formato de audio con el que se quiere capturar el sonido. Al iniciar la grabación desde miniaudio, este se comunica con ALSA indicándole los parámetros seleccionados. Tras recibir esta configuración, ALSA transmite esta información a los controladores del audio, los cuales configuran el ADC para que comience a capturar las muestras en base a los parámetros especificados.

El ADC al recibir las muestras analógicas las convierte a señales digitales. Las señales se transfieren a los controladores y posteriormente a ALSA, el cual al gestionar estos datos se los transmite a miniaudio. Este proceso se mantiene hasta que el usuario finaliza la grabación. Al terminar la grabación, miniaudio notifica a ALSA de que se detenga la captura de audio, a su vez, ALSA comunica esto a los controladores que ordenan al ADC de que se deje de capturar las muestras. Finalmente, miniaudio codifica el archivo .wav con las muestras que ALSA le ha propocionado.

4.5. Reproducción de audio en sistemas GNU Linux

En la siguiente figura 4.5 se representa el proceso de reproducción de audio de la aplicación mediante un diagrama secuencial. La flecha roja indica que ese proceso es continuo hasta que se produzca el aviso de fin de reproducción.

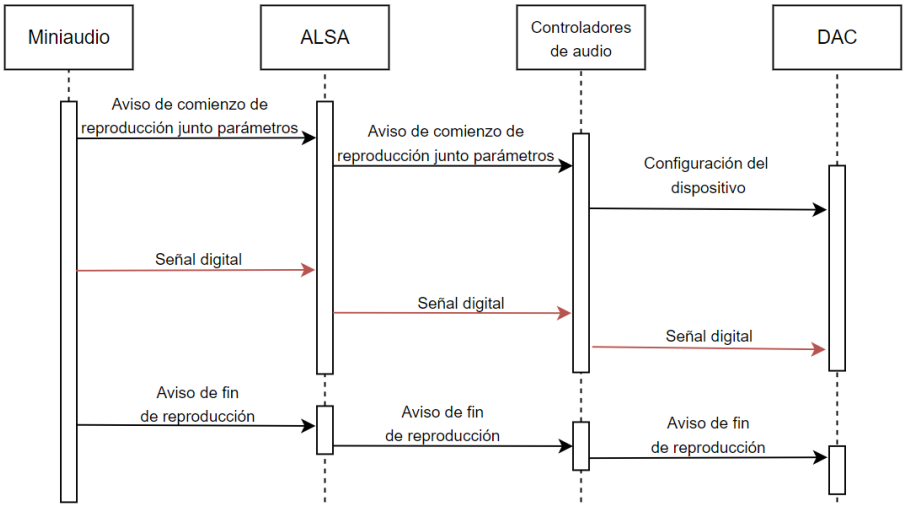

Figura 4.5: Diagrama secuencial del proceso de reproducción

El proceso de reproducción de audio es similar al proceso de captura, solo que en orden inverso. ALSA, al recibir las señales binarias de la aplicación que esté transmitiendo audio, las transmite a los controladores de audio del dispositivo. Los controladores comunican las señales al conversor de señal digital a analógica (DAC). El DAC convierte las muestras digitales a señales analógicas, estas señales generadas se corresponden a la amplitud de la señal de audio en distintos puntos, cada muestra representa un punto. La señal analógica puede contener frecuencias altas, por lo que se utiliza un filtro de paso bajo, eliminando así las frecuencias altas. Tras filtrar las señales, la señal se transfiere a los dispositivos de salida, los cuales mueven el aire,

generando así las ondas analógicas de sonido.

En el marco de este trabajo, para poder reproducir audio a través de miniaudio, se seguiría un orden similar, pero con un paso adicional. Para la reproducción de sonido se ha de crear un decodificador en miniaudio, al que se le especifica la frecuencia, los canales y el formato de audio con el que se quiere reproducir el sonido. Antes de transferir las muestras al DAC, miniaudio manda los parámetros especificados a ALSA y este a los controladores. Los controladores se encargan de configurar el hardware del audio según los parámetros recibidos.

Tras configurar el DAC, el decodificador lee el archivo .wav y pasa las muestras binarias a ALSA, el cual se las pasa a los controladores y estos al DAC. El DAC convierte las señales digitales a analógicas. Las señales analógicas son pasadas por el filtro de paso bajo para eliminar las frecuencias altas y tras eso, las señales se transfieren al dispositivo de salida para que este las reproduzca como sonido.

4.6. Transformada rápida de Fourier

La transformada rápida de Fourier (FFT) es un algoritmo que calcula de manera eficiente la Transformada Discreta de Fourier (DFT). La DFT convierte una secuencia de señales discretas en una secuencia de valores en la frecuencia. Sin embargo, la DFT cuenta con un problema: su alta complejidad computacional, siendo esta $O(N^2)$. Esto se debe a que para cada muestra se deben realizar diversos cálculos. En consecuencia, al tener una gran cantidad de muestras, la DFT puede tardar mucho tiempo en realizar los cálculos o consumir muchos recursos del equipo que realice estos cálculos.

Debido a estas limitaciones, surgió la transformada rápida de Fourier, un algoritmo que realiza los cálculos de la DFT de manera más eficiente. En vez de realizar múltiples cálculos por cada muestra, el algoritmo utiliza un enfoque de "divide y vencerás", dividiendo la señal en otras más pequeñas y realizando la DFT sobre ellas.

En el contexto de esta aplicación, miniaudio, al recibir las muestras (generadas por la captura o por la reproducción), tiene un método de callback en el cual almacena las muestras en un array para posteriormente realizar la FFT sobre ellas. Tras la normalización y el suavizado de las frecuencias, para una visualización más clara, se representan las frecuencias. Como se mencionó anteriormente, la realización de la transformada de Fourier fue un objetivo adicional que se planteó bastante avanzado en el proyecto. Debido a eso, se realiza una transformada bastante básica.

Para realizar la transformada, se ha realizado una implementación propia. Suponiendo estas páginas una gran ayuda para entender el funcionamiento de los conceptos [30] [31]. El código de la implementación es el siguiente:

```
1  void fft(float in[], size_t stride, float complex out[], size_t n)
2  {
3      assert(n>0);
4
5      if(n==1){
6          out[0] = in[0] + I*in[0];
7          return;
8      }
9
10     fft(in, stride*2, out, n/2);
11     fft(in + stride, stride*2, out + n/2, n/2);
12
13     for(size_t k = 0; k < n/2 ; ++k){
14         float t = (float)k/ (n/2);
15         float complex v = cexp(-2*I*PI*t)*out[k + n/2];
16         float complex e = out[k];
17         out[k] = e + v;
18         out[k + n/2] = e -v;
19     }
20 }
```

El llamamiento a la función se realiza con los siguientes parámetros:

- **float in[ ]**: El array que contiene las muestrasde la señal de audio.
- **size_t stride**: Indica el paso entre los elementos sucesivos que queremos considerar.
- **float out[ ]**: El array de salida donde se almacenarán los resultados de la FFT
- **size_t n**: El número de puntos de la FFT. A mayor cantidad de puntos, más precisa es la representación. Pero también aumenta el coste computacional.

Como se puede observar, el algoritmo sigue el enfoque de "divide y vencerás". Primero, para asegurarse de no cometer ningún error, se realiza un *assert* en el que se verifica que la variable de tamaño sea mayor que cero. Posteriormente, se realizan dos llamadas recursivas, en las que se procesan los elementos pares por un lado, y los impares por el otro. Estas llamadas recursivas se realizan hasta que el tamaño sea uno, donde se procesa la muestra y se convierte a un número complejo.

Finalmente, al haber procesado todas las muestras, los resultados se combinan y procesan para obtener la Transformada rápida de Fourier al completo. La estructura recursiva del código y la combinación de los resultados son esenciales para la eficiencia de la transformada. De esta manera, se reduce la complejidad del cálculo de $O(N^2)$ a $O(n * log(n))$.

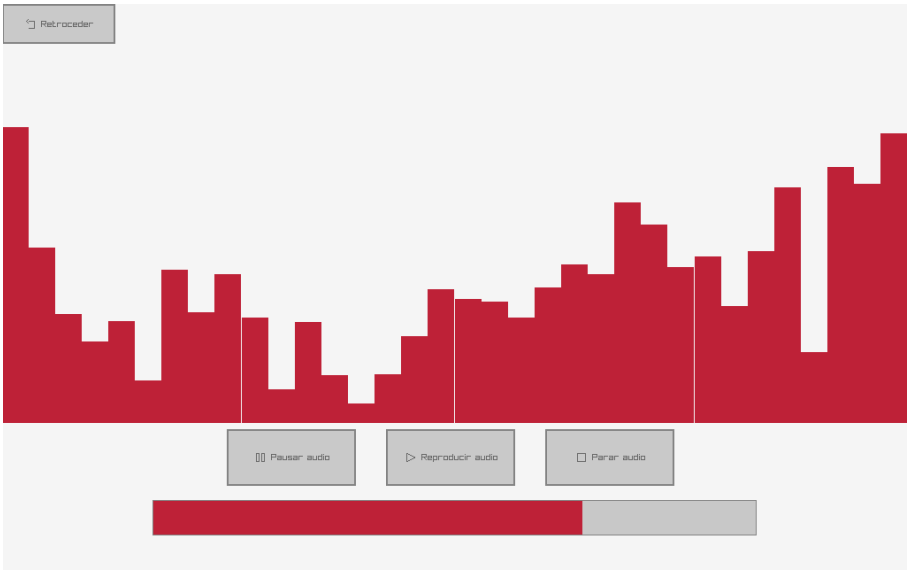

Figura 4.6: Visualización de las frecuencias en la pantalla de reproducción

4.7. Conversión de formato de archivo de audio

La conversión de archivos WAVE a formato texto es sencilla. Como se ha visto anteriormente, los archivos .wav están únicamente compuestos de datos binarios, los cuales pueden convertirse fácilmente a hexadecimal en C. Con los datos en hexadecimal, se pueden pasar al formato .txt para poder editar fácilmente el archivo WAVE.

En la aplicación para realizar esto, se realizan varios *fread*, uno por cada parte de la cabecera del archivo .wav y posteriormente, se hace otro *fread* de cada muestra del archivo WAVE y se almacena en un buffer la muestra. De este modo, el archivo de texto contendrá la información en hexadecimal.

```
1 while(1){
2     read = fread(buffer2, sizeof(buffer2), 1, fWav);
3     if(read == 0)
4         break;
5
6     number = buffer2[0] | (buffer2[1] << 8);
7     buffer[i++] = (short int)number;
8 }
```

Para convertir un archivo de texto en uno de tipo WAVE, se emplea el mismo método, solo que esta vez el archivo del que se realizará la lectura será el de texto en vez del de audio.

4.8. Limitaciones y problemas encontrados

Durante el desarrollo del proyecto se encontraron diversas limitaciones, algunas relacionadas con miniaudio y otras con raylib.

La primera limitación, ya mencionada anteriormente, surge con el uso de miniaudio. Esto se debe a que miniaudio trabaja bien con archivos .wav, pero con los otros dos formatos que también debería ser capaz de manejar, como son las extensiones .mp3 y .flac, tiene problemas para extraer las muestras de la onda de sonido de los archivos. Debido a esto, no se podía realizar la transformada de Fourier con estas dos extensiones. Por lo tanto, se limitó la aplicación para que solo se pudieran utilizar archivos WAVE.

Otro problema que se encontró en este trabajo se debió a raylib. Esto ocurrió porque originalmente se realizó el proyecto utilizando raudio, un módulo de raylib que incluye miniaudio y múltiples librerías más. Sin embargo, al intentar implementar la funcionalidad de captura de audio, surgió un problema: raudio redefinía los formatos de codificación, lo que provocaba que, al grabar, el audio no se guardara correctamente. Debido a este inconveniente, se tuvo que modificar el proyecto para que pasara a utilizar directamente miniaudio, sin inicializar el módulo de raudio.

Capítulo 5

Impacto del trabajo

El desarrollo de este trabajo tiene potencial para contribuir al cuarto objetivo de desarrollo sostenible: Educación de calidad. Esto se debe a dos factores. Con el contenido de esta memoria, uno puede aprender diversos conceptos relacionados con el audio. Tanto analógico como digital.

Además, con la aplicación que se ha implementado. También se puede aprender sobre estos conceptos. Así mismo, mediante el código fuente de la aplicación también es posible aprender sobre el desarrollo de aplicaciones implementadas en lenguaje C destinadas a sistemas GNU/Linux.

Capítulo 6

Resultados y conclusiones

6.1. Resultados

Como se ha mencionado previamente en la introducción, este trabajo trata de crear una aplicación orientada a sistemas GNU/Linux enfocada en tanto la captura como la visualización de señales de audio. Haciendo uso de tanto librerías nativas de C como de terceros.

En este proyecto se ha logrado cumplir los objetivos iniciales satisfactoriamente. Se ha realizado la implementación de una aplicación desarrollada en lenguaje C en un entorno GNU/Linux. Esta aplicación permite tanto la captura como la reproducción de señales de audio.

Además, se ha realizado un estudio a fondo de los conceptos referentes a los archivos, señales de audio, formatos y dispositivos asociados. También se han revisado iniciativas relacionadas tanto en lenguaje C como en otros lenguajes, suponiendo algunas de estas iniciativas una gran ayuda para entender el funcionamiento de los conceptos previamente mencionados.

También se ha elaborado una documentación asociada para entender el funcionamiento de la aplicación, la cual se encuentra desarrollada en el Anexo.

Por último, los objetivos adicionales también han sido cumplidos. Se ha conseguido visualizar correctamente las señales de audio mediante la transformada de Fourier. También se ha logrado el cambio de formato de los archivos de las señales de audio y la visualización de la información del archivo. Todo esto puede realizarse tanto a través de una interfaz de usuario como desde la consola.

6.2. Conclusiones personales

Durante el desarrollo de este proyecto he descubierto muchas cosas que antes no sabía cómo funcionaban. Nunca me había detenido a pensar cómo funcionaba el micrófono de mi ordenador o cómo el ordenador reproducía toda la música que he estado escuchando durante toda la realización de este trabajo. Sin embargo, debo decir que me ha resultado muy interesante aprender cómo funciona todo esto.

También, cuando empecé este trabajo, no me gustaba mucho C; me parecía un lenguaje engorroso en el cual para hacer una cosa simple necesitabas varios pasos. Pero actualmente pienso lo contrario, que nunca me molesté en aprender cómo funcionaba este lenguaje. C me parece un lenguaje muy versátil en el que se pueden hacer tanto cosas de alto como de bajo nivel, y que una vez se aprenden cosas como realizar un manejo de la memoria correcto, por ejemplo, es cuando uno descubre el potencial de este lenguaje.

En resumen, termino este TFG sabiendo mucho más de lo que sabía antes y muy satisfecho con el trabajo que he realizado.

6.3. Trabajo futuro

Como plan de futuro, se ha sugerido el desarrollo de una nueva funcionalidad. Esta consistirá en la posibilidad del cambio de frecuencia de muestreo de los archivos audio. Para esto, es necesario realizar un estudio de los distintos algoritmos de remuestreo y cómo implementarlos en lenguaje C de una manera correcta. Esto, con el fin de que el audio remuestreado represente fielmente al audio original y no se produzca aliasing.

Bibliografía

[1] M. Shamim Hossain. «Patient State Recognition System for Healthcare Using Speech and Facial Expressions». En: *Journal of Medical Systems* 40.272 (2016). DOI: 10.1007/s10916-016-0627-x. URL: https://doi.org/10.1007/s10916-016-0627-x.

[2] Wikipedia. *GNU/Linux:* URL: https://es.wikipedia.org/wiki/GNU/Linux (visitado 23-05-2024).

[3] Alexander S. Gillis. *What is GNU/Linux?:* 2022-05. URL: https://www.techtarget.com/searchdatacenter/definition/GNU-Linux (visitado 23-05-2024).

[4] Thomas Alsop. *Distribution of the 500 most powerful supercomputers worldwide from 2017 to 2023, by operating system:* 2024-03-07. URL: https://www.statista.com/statistics/565080/distribution-of-leading-supercomputers-worldwide-by-operating-system-family/#:~:text=As%20of%20June%202023%2C%20of,used%20the%20CentOS%20operating%20system. (visitado 23-05-2024).

[5] Wikipedia. *Unix:* URL: https://es.wikipedia.org/wiki/Unix (visitado 23-05-2024).

[6] Wikipedia. *C (lenguaje de programación):* URL: https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n) (visitado 23-05-2024).

[7] *Lista de lenguajes populares:* URL: https://www.orientsoftware.com/blog/most-popular-programming-languages/ (visitado 23-05-2024).

[8] Joe Warminsky. *Memory-Related Software Bugs*. 2024. URL: https://therecord.media/memory-related-software-bugs-white-house-code-report-oncd (visitado 23-05-2024).

[9] *Adobe Audition:* URL: https://www.adobe.com/products/audition.html (visitado 23-05-2024).

[10] *Pro Tools:* URL: https://www.avid.com/pro-tools (visitado 23-05-2024).

[11] *Audacity:* URL: https://www.audacityteam.org/ (visitado 23-05-2024).

[12] *PulseAudio:* URL: https://www.freedesktop.org/wiki/Software/PulseAudio/ (visitado 23-05-2024).

[13] *FFmpeg:* URL: https://ffmpeg.org/ (visitado 23-05-2024).

[14] *Miniaudio:* URL: https://miniaud.io/index.html (visitado 23-05-2024).

[15] *dr_wav:* URL: https://github.com/mackron/dr_libs/blob/master/dr_wav.h (visitado 23-05-2024).

[16] *Musializer:* URL: https://github.com/tsoding/musializer (visitado 23-05-2024).

[17] *GTK:* URL: https://www.gtk.org/ (visitado 23-05-2024).

[18] *raylib:* URL: https://www.raylib.com/ (visitado 23-05-2024).

[19] Gus Berry y Lo Boutillette. *Choosing the Best Audio Format for Your Projects*. URL: https://www.adobe.com/creativecloud/video/discover/best-audio-format.html (visitado 10-05-2024).
[20] Wikipedia. *Audio file format:* URL: https://en.wikipedia.org/wiki/Audio_file_format (visitado 23-05-2024).
[21] *Ranking de distribuciones Linux más utilizadas:* URL: https://distrowatch.com/dwres.php?resource=popularity (visitado 23-05-2024).
[22] *PortAudio:* URL: https://www.portaudio.com/ (visitado 23-05-2024).
[23] *OpenAL:* URL: https://www.openal.org/ (visitado 23-05-2024).
[24] *Miniaudio ejemplos:* URL: https://miniaud.io/docs/examples/index.html (visitado 23-05-2024).
[25] *raygui:* URL: https://github.com/raysan5/raygui (visitado 23-05-2024).
[26] *WAVE PCM soundfile format:* URL: http://soundfile.sapp.org/doc/WaveFormat/ (visitado 23-05-2024).
[27] Wikipedia. *Resource Interchange File Format:* URL: https://es.wikipedia.org/wiki/Resource_Interchange_File_Format (visitado 23-05-2024).
[28] Wikipedia. *Hearing range:* URL: https://en.wikipedia.org/wiki/Hearing_range (visitado 23-05-2024).
[29] *Advanced Linux Sound Architecture (ALSA):* URL: https://www.alsa-project.org/wiki/Main_Page (visitado 23-05-2024).
[30] Wikipedia. *Formula de Euler:* URL: https://en.wikipedia.org/wiki/Euler%27s_formula (visitado 01-06-2024).
[31] Wikipedia. *Transformada rápida de Fourier:* URL: https://en.wikipedia.org/wiki/Fast_Fourier_transform (visitado 01-06-2024).
[32] *Instalación de raylib en GNU Linux:* URL: https://github.com/raysan5/raylib/wiki/Working-on-GNU-Linux (visitado 23-05-2024).

Apéndice A

Anexo

A.1. Manual de usuario

La estructura de las carpetas del proyecto es la siguiente A.1:

- La carpeta **lib** contiene las librerías que la aplicación necesita para poder ejecutarse.
- La carpeta **songs**, a pesar de estar vacía, ha sido creada para tener un lugar en el que poder acceder a los archivos de audio facilmente.
- La carpeta **src** contiene el código del programa.
- Por último, la carpeta **styles** contiene distintos estilos visuales para la aplicación compatibles con raylib.
- El ejecutable **api** es el ejecutable de la aplicación.
- El archivo **build.sh** es un archivo ejecutable con el que se puede compilar el programa.
- Finalmente, el archivo **Readme.txt** contiene un pequeño manual para el usuario.

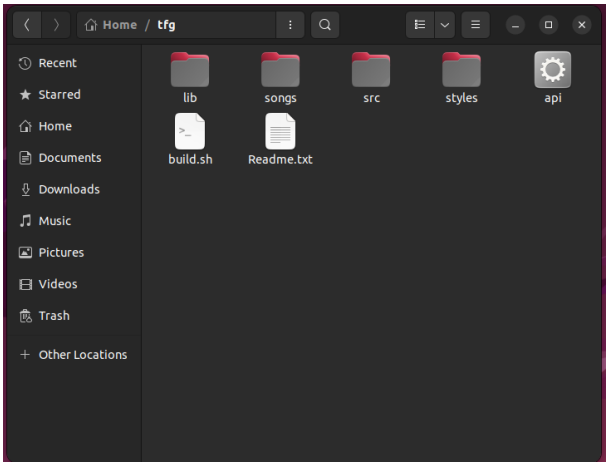

Figura A.1: Estructura carpetas

Para poder ejecutar la aplicación es necesario tener raylib instalado en el equipo. Para realizar la instalación consultar el siguiente enlace: [32]. Para acceder a la aplicación con interfaz de usuario ejecutar el comando "*./api*" desde la carpeta */tfg*. En caso de querer acceder a la aplicación en modo consola ejecutar "*./api konsole*". Si se necesita ayudar ejecutar el comando "*./api –help*".

Al acceder a la aplicación con interfaz de usuario se presenta la pantalla inicial de la aplicación. Para poder tanto reproducir como para capturar se necesita seleccionar antes un archivo, sino saltará un aviso de selección de archivo. Para reproducir seleccionar el botón debajo del de reproducción. En caso de haber seleccionado un archivo .wav se mostrará este debajo y se permitirá la reproducción. Para la captura se sigue el mismo funcionamiento, únicamente que ahí se ha de escribir el archivo que se quiera crear. Por último también se puede acceder a los ajustes de la aplicación como pulsar la flecha a la derecha y acceder a más funciones.

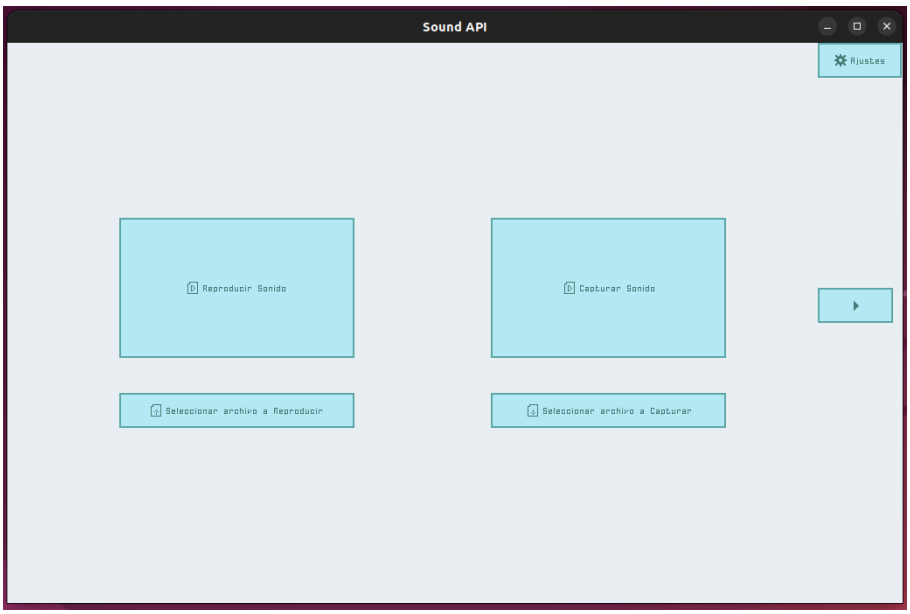

Figura A.2: Pantalla inicial aplicación

En la pantalla de reproducción se presentan las opciones de reproducir el audio, pausar el audio y pararlo. A su vez, también se puede parar y reanudar el audio con la tecla de espacio. También se encuentra una barra debajo de los botones que representa el tiempo de reproducción del audio. En la pantalla de grabación hay un botón de grabación y también se puede pulsar la tecla de espacio para iniciar y parar la captura de audio.

En la sección de ajustes se pueden realizar diversos cambios a la aplicación. Se puede cambiar el estilo visual del programa, la frecuencia de muestreo con la que reproducirá y se capturará el audio. Por último también se puede seleccionar si ver o no las frecuencias y el color de las barras de las frecuencias.

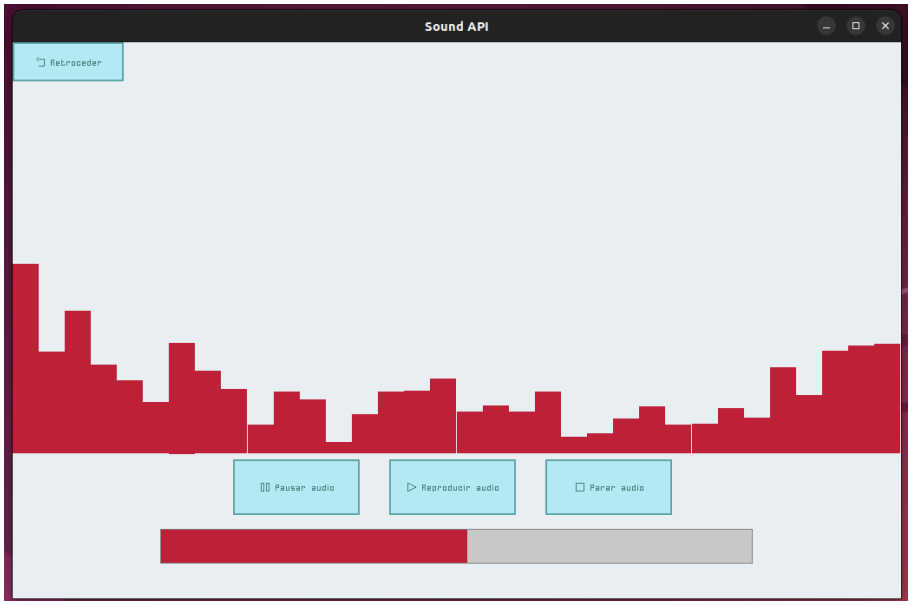

Figura A.3: Pantalla de reproducción

Por último si se pulsa sobre la flecha a la derecha se accede a más funcionalidades de la aplicación. Pulsando al botón de selección de archivo, se puede elegir un fichero. El archivo puede ser tanto .txt como .wav. Se puede seleccionar el botón de conversión de archivo tanto con un .txt como con un .wav seleccionado. El programa detectará que extensión de archivo es y lo convertirá a la contraria. Para ver la información de un archivo también se pueden tener ambas extensiones. Por último, para cambiar la frecuencia de muestreo se ha de tener seleccionado un archivo del tipo .wav, el programa doblará o pondrá a la mitad la frecuencia de muestreo del archivo seleccionado, el nuevo archivo se guardará en la carpeta *songs* a nombre de upsample.wav o downsample.wav.

Finalmente, al ejecutar el programa en modo consola existen los siguientes comandos:

- **Comando man**: Al ejecutar este comando se detallarán el resto de los comandos disponibles en la aplicación.
- **Comando play**: Al ejecutar este comando se pedirá al usuario la ruta del archivo que se desea reproducir y la frecuencia de muestreo con la que se reproducirá el archivo.
- **Comando record**: Al ejecutar este comando se pedirá al usuario la ruta del archivo en la que se desea guardar la captura de sonido y la frecuencia de muestreo con la que se capturará el archivo.
- **Comando convert**: Al ejecutar este comando se pedirá al usuario la ruta del archivo del archivo que se desea convertir si el archivo seleccionado tiene extensión .wav se convertirá a .txt y viceversa.

- **Comando info**: Al ejecutar este comando se pedirá al usuario la ruta del archivo del archivo del que se desea saber su información, si no ocurre ningún error, se mostrará la información del archivo seleccionado.
- **Comando upsample**:Al ejecutar este comando se pedirá al usuario la ruta del archivo del archivo del que se desea aumentar su frecuencia de muestreo, si no ocurre ningún error, se creará un archivo llamado upsample.wav en la carpeta songs.
- **Comando downsample**:Al ejecutar este comando se pedirá al usuario la ruta del archivo del archivo del que se desea dividir por dos su frecuencia de muestreo, si no ocurre ningún error, se creará un archivo llamado downsample.wav en la carpeta songs.

A.2. Informe de originalidad

El informe de originalidad proporcionado por la herramienta Turnitin se encuentra disponible a partir de la siguiente página.

TFG.pdf

ORIGINALITY REPORT

11%	10%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Universidad Politécnica de Madrid Student Paper	2%
2	oa.upm.es Internet Source	1%
3	Submitted to Universidad TecMilenio Student Paper	1%
4	www.ptolomeo.unam.mx:8080 Internet Source	<1%
5	Submitted to Infile Student Paper	<1%
6	Mayurakshi Jana, Suparna Biswas. "Intelligent and smart enabling technologies in advanced applications: recent trends", Elsevier BV, 2021 Publication	<1%
7	es.wikipedia.org Internet Source	<1%
8	hdl.handle.net Internet Source	<1%

9	Submitted to University Of Tasmania Student Paper	<1 %
10	en.wikipedia.org Internet Source	<1 %
11	gsyc.urjc.es Internet Source	<1 %
12	uvadoc.uva.es Internet Source	<1 %
13	www.idg.es Internet Source	<1 %
14	prezi.com Internet Source	<1 %
15	pvs.uad.lviv.ua Internet Source	<1 %
16	www.chismesmundo.com Internet Source	<1 %
17	issuu.com Internet Source	<1 %
18	www.wirelessargentina.com.ar Internet Source	<1 %
19	www.statista.com Internet Source	<1 %
20	upcommons.upc.edu Internet Source	<1 %

21	www.06gl.com Internet Source	<1 %
22	www.american.edu.co Internet Source	<1 %
23	www.doctorproaudio.com Internet Source	<1 %
24	www.hostinger.co Internet Source	<1 %
25	ela.kpi.ua Internet Source	<1 %
26	www.neoteo.com Internet Source	<1 %
27	www.audio-digital.net Internet Source	<1 %
28	api.openalex.org Internet Source	<1 %
29	catalonica.bnc.cat Internet Source	<1 %
30	event.nchc.org.tw Internet Source	<1 %
31	news.pntic.mec.es Internet Source	<1 %
32	pypi.org Internet Source	<1 %

33	www.adobe.com Internet Source	<1 %
34	www.comimsa.com.mx Internet Source	<1 %
35	www.sigmados.com Internet Source	<1 %
36	bluestacks.es.downloadastro.com Internet Source	<1 %
37	evnuir.vnu.edu.ua Internet Source	<1 %
38	securityinabox.org Internet Source	<1 %
39	www.casarramona.com Internet Source	<1 %
40	catarina.udlap.mx Internet Source	<1 %
41	ftp.snt.utwente.nl Internet Source	<1 %
42	libros.catedu.es Internet Source	<1 %
43	patents.google.com Internet Source	<1 %
44	tecnoticias.net Internet Source	<1 %

45	www.colkinechile.cl Internet Source	<1 %
46	www.documents.philips.com Internet Source	<1 %
47	www.researchgate.net Internet Source	<1 %
48	www.timetoast.com Internet Source	<1 %
49	zaguan.unizar.es Internet Source	<1 %
50	es.unionpedia.org Internet Source	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Mon Jun 03 15:32:44 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)