Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Ingeniería en Computación

Proyecto Terminal de Ingeniería en Computación

Codificación Concurrrente de Audio en Arquitecturas Multi-Núcleo

Proyecto que presenta:

Angel González Méndez

para obtener el titulo de:

Ingeniero en Computación

Asesor del Proyecto:

M. en C. Oscar Alvarado Nava

México, D.F. **Agosto de 2010**

Resumen

MPEG-1 Audio Layer 3, mas conocido como *MP3* es un formato de audio digital comprimido con perdidas, la codificación a este formato se lleva acabo mediante la eliminación de información que no es perceptible por el oído humano, una codificación estandar de este formato mantiene la calidad de un CD de audio, con la diferencia de que su tamaño es aproximadamente de un diez porciento con respecto a un formato de audio sin perdidas, el fomato *MP3* es muy utilizado en la web debido a su calidad y su reducido tamaño.

Por otro lado la tecnología crece rápidamente y esto se ve reflejado en la arquitectura de las computadoras con el auge de los procesadores multi-núcleo, gracias a esto ha aumentado la difusión del procesamiento paralelo. Aún con la difusión de este tipo de procesamiento, la mayor parte de las aplicaciones están programadas de forma secuencial esto hace que no se aprovechen las nuevas arquitecturas multi-núcleo, dejando a un lado la posibilidad de aumentar significativamente el rendimiento de las mismas.

Por medio de la programación paralela es posible desarrollar aplicaciones con las cuales se obtiene un rendimiento mayor que si se programan de forma secuencial, además de aprovechar los recursos de las nuevas arquitecturas. Para llevar acabo este tipo de programación se requiere de un análisis previo de la aplicación a paralelizar, esto es para localizar la tarea que es mas exigente en cuestión de procesamineto, apartir del análisis realizado se diseña un algoritmo para que dicha tarea se procese de forma paralela, tomando en cuenta que no debe existir dependencia de datos ya que de exisitir se generarían condiciones de competencia que se deben resolver, ya que de no ser así el resultado puede ser incierto. El procesamiento paralelo en este tipo de arquitecturas se maneja a través de hilos(*threads*).

En el presente documento se presenta el desarrollo de una aplicación, la cual realizara la codificación de un archivo de audio *WAV* a *MP3* de forma paralela, esto a través de la programación multi-hilo, dicha aplicación disminuirá notablemente el tiempo de codificación respecto a la solución secuencial.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivaciones

Actualmente la tecnología crece rápidamente y esto se ve reflejado en la arquitectura de las computadoras con el auge de los procesadores multi-núcleo. Esto ha aumentado la difusión del procesamiento concurrente y paralelo.

Con el paso del tiempo los sistemas de memoria compartida se van integrando con más núcleos en un solo chip. Al tener varios núcleos en un solo chip las necesidades de cómputo de numerosas aplicaciones obligan a desarrollar software que saque ventaja de las arquitecturas multi-núcleo. No obstante, para utilizar este tipo de arquitecturas de forma eficiente es necesaria la programación multi-hilo.

La codificación es ante todo, la conversión de un sistema de datos a otro distinto; la codificación puede llevar a una compresión. De ello se desprende que la información resultante es equivalente a la información de origen. Un modo sencillo de entender esto es verlo a través de los idiomas, en el ejemplo siguiente: home = hogar, podemos entender que hemos cambiado una información de un sistema (inglés) a otro sistema (español) y que esencialmente la información sigue siendo la misma. La razón de la codificación está justificada por las operaciones que se necesite realizar con posterioridad.

Los métodos de codificación de audio que existen en la actualidad se basan en algoritmos de compresión y en codificación multicanal, los algoritmos de compresión de audio se fundamentan en aspectos perceptibles al oído humano.

WAV [12][13], apócope de *WAVEform audio format*, es un formato de audio digital sin compresión de datos. Al ser un archivo sin compresión es muy grande en tamaño y contiene información que no es perceptible al oído humano.

MPEG-1 Audio Layer 3 [14][15][22][19], más conocido como *MP3*, es un formato

de audio digital comprimido con pérdida, desarrollado por el Grupo de Expertos en Imágenes en Movimiento (MPEG [15]) para formar parte de la versión 1 (y posteriormente ampliado en la versión 2) del formato de vídeo MPEG.

La codificación de un archivo *WAV* a *MP3* se lleva a cabo mediante un algoritmo de codificación/compresión fundamentado en aspectos perceptibles al oído humano, lo cual hace que de un archivo más grande se genere uno más pequeño, con pérdida de calidad. Aun así, esta pérdida no es perceptible al oído humano.

La computación paralela emplea elementos de procesamiento múltiple simultáneo para resolver un problema. Esto se logra dividiendo un problema en partes independientes, de tal manera que cada elemento del procesamiento pueda ejecutar su parte del algoritmo a la vez que los demás. Los elementos de procesamiento pueden ser diversos e incluir recursos tales como un sistema de cómputo multi-procesador o multi-núcleo, varios sistemas de cómputo en red, hardware especializado o una combinación de los anteriores.

1.2. Objetivos

En este reporte, se presenta la implementación de un aplicación para la codificación de audio digital *WAV* a *MP3* , la implementación de la mima se llevará a cabo con programación multi-hilo bajo arquitecturas multi-núcleo. El objetivo primordial de este proyecto es el desarrollo de una aplicación multi-hilo para la codificación de audio *WAV* a *MP3* cuyo desempeño sea mejor, lo cual disminuirá el tiempo de codificación respecto a la codificación secuencial.

Para alcanzar el objetivo principal del proyecto, se dividio en los siguientes objetivos particulares:

- **Diseño e Implementación del módulo de reconocimiento.** Permitirá determinar si el archivo a codificar es un archivo *WAV*, esto se realizará a través del análisis del encabezado del archivo. Si el archivo examinado resulta ser *WAV*, se guardará la información del encabezado.

- **Diseñar e Implementación del módulo de división.** Tomando en cuenta la información del encabezado del archivo *WAV*, este módulo verifica cuántos núcleos contiene la arquitectura, al saber cuantos núcleos tiene la arquitectura se podrá elegir el número de hilos a utilizar y con base en esto se calcula la posición desde la cual cada hilo leera el archivo *WAV*. Cabe mencionar que lo ideal será que el número de hilos sea igual al número de núcleos.

- **Diseño e Implementación del módulo de codificación.** Se encargará de la codificación de audio *WAV* a *MP3* de forma concurrente. Esto se logra creando

un hilo para cada parte a codificar, es decir cada hilo leera desde una posición dada por el módulo anterior; por cada hilo se generara un archivo *MP3*.

- **Diseño e Implementación del módulo de unión.** Se encarga de unir los archivos *MP3* generados en el módulo de codificación en un nuevo archivo *MP3*.
- **Diseño e Implementación de una GUI**. A través de esta se integraran los módulos desarrollados
- **Validación y experimentos.** Una vez que los módulos son integrados, se verificó inicialmente que la aplicación desarrollada realice la codificación de forma correcta. Validada la aplicación, se llevaron a cabo experimentos similares tanto con la aplicación desarrollada y una aplicación que realiza la codificación de forma secuencial.

1.3. Organización del Reporte

El Reporte está organizado de la siguiente manera: En el Capítulo 2 se presenta una introducción al formato de audio *MP3*[22], mostrando como se construye dicho formato. Se lleva a cabo en el mismo capítulo una descripción de la codificación *MP3*[14][15][22], enumerando los pasos a seguir para llevarla a cabo, también se muestra una descripción del codificador a paralelizar LAME[7].

En el Capítulo 3 se muestra el desarrollo del proyecto, comenzando con la descripción de la metodología para paralelizar el codificador seleccionado, una vez descrita la metodología se describe el diseño e implementación de los módulos que conformaran la aplicación mostrando el pseudocódigo de los algoritmos desarrollados, al final del capítulo se describe el desarrollo de la GUI para la aplicación.

En el Capítulo 4 se presenta el análisis de los resultados obtenidos entre las dos soluciones:

1. Codificación secuencial utilizando el codificador LAME[7]
2. Codificación concurrente utilizando la aplicación desarrollada

Finalmente, las conclusiones se presentan en el Capítulo 5.

Capítulo 2

Codificación MP3

2.1. Introducción

MPEG[15], es el nombre de un grupo de trabajo creado bajo la dirección conjunta de la Organización Internacional de Normalización / Comisión Electrotécnica Internacional (ISO / IEC)[16][17], que tiene por objetivo crear normas para el vídeo digital y compresión de audio. Más precisamente, MPEG define la sintaxis de formatos de audio y video que necesitan velocidades de transmisión baja, así como las operaciones que llevarán a cabo los decodificadores. Los algoritmos utilizados por los codificadores no están definidos por MPEG. Este codificador autoriza la mejora continua, así como su adaptación a aplicaciones específicas, sin que resulte necesaria cualquier redefinición de la disposición de datos. Junto a la codificación de audio y vídeo MPEG define métodos con el objetivo de probar la conformidad con los estándares de formatos y decodificadores además de publicar informes técnicos.

MP3[14][15][22][19], abreviatura de MPEG-1/MPEG-2 Layer 3, es un formato para almacenar audio digital. Se utiliza un tipo avanzado de compresión de audio lo que reduce el tamaño del archivo con poca reducción en la calidad de audio. *MP3*, se utiliza en aplicaciones de software, reproductores digitales de audio, dispositivos de equipo de sonido y la distribución de música por Internet, pero también se utiliza para otros fines, como las transmisiones de audio digital en tiempo real a través de RDSI[18]. El formato *MP3* (MPEG-1/MPEG-2 Layer 3), es un estándar ISO desde 1993.

El Instituto Fraunhofer[19] ha sido el principal desarrollador del formato MPEG Layer-3. El estándar *MP3* que ha sido aprobado se basa principalmente en su trabajo, el cual ha protegido por varias patentes. El Instituto Fraunhofer y Thomson Multimedia[20] (también conocido como RCA) decidieron unir sus patentes sobre este formato con el fin de crear una cartera de patentes conjunta y pedir regalías por el uso de esta cartera.

Cabe mencionar que las condiciones de uso de los formatos *WAV* y *MP3*, estipulan que pueden utilizarce libremente siempre y cuando no se lucre con el software desarrollado de lo contrario hay que pagar regalias.

2.2. Codificación de audio MPEG Layer-III

¿Cómo funciona MPEG?

Un compresor de audio MPEG se basa en un sistema de codificación perceptual, durante una codificación perceptual, el codec no trata de mantener una señal absolutamente idéntica después de la codificación de las señales originales, pero su objetivo es asegurar que la señal de salida parezca idéntica para el oído humano. El primer efecto psicoacústico que usa la codificación perceptual es el efecto de ocultación, basado en el hecho de que algunas partes de la señal debido al funcionamiento del sistema auditivo humano no son audibles. Para ser capaces de suprimir esta señal, el codificador integra un modelo psicoacústico tratando de imitar el comportamiento del oído humano. Este modelo psicoacústico analiza la señal de entrada en varios bloques consecutivos y determina para cada bloque el espectro de la señal. A continuación, se modelan las propiedades de enmascaramiento del sistema auditivo humano, y las estimaciones del nivel audible mínimo. Durante su cuantificación y la fase de codificación, el codificador intenta asignar el número de bits a fin de respetar las propiedades de enmascaramiento, así como el tamaño del tipo de datos autorizados.

Etapas

Hay que distinguir dos puntos. En primer lugar, MPEG trabaja por etapas. Estas etapas son normalmente denota en numeración arábiga (MPEG-1, MPEG-2, MPEG-4). La primera etapa, el MPEG-1, establece la codificación de sonidos estéreo y monofónico, a las frecuencias de uso general para una calidad de audio alta son 48, 44.1 y 32 KHz. La segunda etapa consta de dos maneras diferentes de trabajo. La primera es la extensión a las frecuencias más débiles de grabación, proporcionando una mejor calidad de resonancia (menos de 64 Kbits/s para una señal monofónica). La segunda manera es la extensión sonidos incluyendo varias voces. MPEG-1 y MPEG-2 tienen una estructura de tres capas(*layers*). Cada capa representa una familia de algoritmos de codificación. Estas capas se denotan con números romanos (Layer I, Layer II, Layer III).

Capas (*Layers*)

Las diferentes capas se han definido en el estándar y cada capa tiene sus propias ventajas.Además, la complejidad aumenta al pasar de la *Layer* I de la *Layer* III.

1. *Layer* I pose la menor complejidad y está específicamente dirigido a aplicaciones donde la complejidad del codificador juega un papel importante

2. *Layer* II requiere un codificador más complejo, así como un decodificador más complejo. En comparación con *Layer* I, este es capaz de suprimir más redundancia en la señal y se aplica el modelo psicoacústico de una manera más eficiente

3. *Layer* III tiene un complejidad cada vez mayor y está dirigido a aplicaciones que requieran la menor tasa de datos, por la supresión de la señal redundante mejora la extracción de frecuencias débilmente audibles, utilizando su filtro.

Modo de Operación

MPEG-1 Audio funciona tanto para señales mono y estéreo. Una técnica llamada codificación *joint stereo* se puede utilizar para lograr una codificación más eficiente combinando los canales izquierdo y derecho de una señal de audio estereofónico. Este último es especialmente útil para bajar las tasas de bits. Los modos de funcionamiento son:

- canal único
- doble canal
- estéreo
- *joint stereo*

El umbral de audición mínima

El umbral de audición mínima del oído humano no es lineal. Se representa, de acuerdo a la ley de Fletcher-Munson, por una curva de excavado entre 2 y 5 KHz. No son necesarios sonidos situados debajo de este umbral, porque no los perciben el oído humano.

El efecto de enmascaramiento

Este sistema se basa en las propiedades de enmascaramiento del oído humano por ejemplo: cuando se mira al sol y si un pájaro pasa por delante, no lo ve porque la luz del sol es demasiado predominante. En audio, es similar. Durante los sonidos fuertes, usted no escucha a los sonidos más débiles. Tomemos como ejemplo una pieza de órgano: cuando el organista no toca, se oye la respiración en las tuberías, y cuando él toca, ya no lo escucha porque está enmascarado.

Por lo tanto, no son necesarios todos los sonidos. Esta es la primera propiedad utilizada por el formato *MP3* para ganar algo de espacio. Para ello el codificador *MP3* utiliza un modelo psicoacústico simulando el comportamiento del oído humano.

Bytes de depósito

A menudo, algunos pasajes de una pieza musical no pueden ser codificados a una frecuencia sin que se modifique la calidad del audio. En este caso la codificación *MP3*, utiliza un depósito corto de bytes que actúa como un buffer mediante el uso de la capacidad de los pasajes, que pueden ser codificados a una frecuencia inferior en un flujo determinado.

La codificación *joint stereo*

En el caso de una señal estereofónica, el formato *MP3* puede utilizar algunas herramientas más, como es la codificación *Joint Stereo* (JS), para reducir aún más el tamaño del archivo comprimido.

En muchos sistemas de alta fidelidad de gama media, estos tienen un único subwoofer, sin embargo por lo general no se tiene la sensación de que el sonido provenga de este, sino de los altavoces satélite. Para frecuencias muy bajas o muy altas, el oído humano no es capaz de localizar el origen espacial de sonidos con una precisión total. El formato mp3 por lo tanto puede (opcionalmente) usar la intensidad estéreo (IS). Algunas frecuencias se registran como una señal monofónica seguida de una información adicional a fin de restablecer un mínimo de especialización.

La codificación Huffman

La codificación *MP3* también utiliza el algoritmo de Huffman. Actúa al final de la compresión para codificar la información, la codificación Huffman no es propiamente un algoritmo de compresión, es un método de codificación.

La codificación Huffman crea códigos de longitud variable en un número entero de bits. Los códigos de Huffman tienen la propiedad de tener un prefijo único, por lo tanto, pueden ser decodificados correctamente a pesar de su longitud variable. El paso de decodificación es muy rápido (a través de una tabla de correspondencia). Este tipo de codificación permite ahorrar en promedio un poco menos del 20 % del espacio.

Es un complemento ideal de la codificación perceptual: durante las grandes voces, la codificación perceptual es muy eficiente porque muchos sonidos son enmascarados o disminuidos, pero poca información es idéntica, por lo que el algoritmo de Huffman es muy pocas veces eficaz. Durante sonidos “puros” son pocos los efectos de ocultación, pero luego Huffman es muy eficiente porque el sonido digitalizado contiene muchos bytes repetitivos, que luego serán sustituidos por el código más corto.

Frecuencia de muestreo

La compresión de audio MPEG trabaja con diferentes frecuencias de muestreo. MPEG-1 define la compresión de audio a 32 KHz, 44.1 KHz y 48 KHz. MPEG-2 se extiende a la mitad de estas frecuencias, es decir, 16 KHz, 22.05 KHz y 24KHz. MPEG-2.5 es el nombre de una ampliación de propiedad del *Layer*-3, desarrollado por Fraunhofer IIS[19], que introduce las frecuencias de muestreo 8 KHz, 11.05 KHz y 12KHz.

Tasa de Bits (*bit-rate*)

MPEG Audio no sólo trabaja en una relación de compresión fija. La selección de la tasa de bits del audio comprimido dentro de ciertos límites, completamente a la izquierda del ejecutor o gestor de un codificador de audio MPEG. Para el *Layer*-3, el estándar define un rango de tasas de bits es de 8 Kbit/s hasta 320 Kbit/s. Por otra parte, los decodificacores de *Layer*-3 deben apoyar el cambio de tasa de bits de un marco de audio combinado con la tecnología de depósito de bits, esto permite que los bits de codificación de tasa variable y constantes se codifiquen en cualquier valor fijo dentro de los límites establecidos por el estándar.

2.3. Algoritmo de Codificación *MP3*

Los párrafos siguientes describen el algoritmo de codificación con *Layer*-3 junto con los bloques básicos de un codificador perceptual. La Figura 2.1 muestra el diagrama de bloques de un codificador típico de MPEG-1 Layer-3.

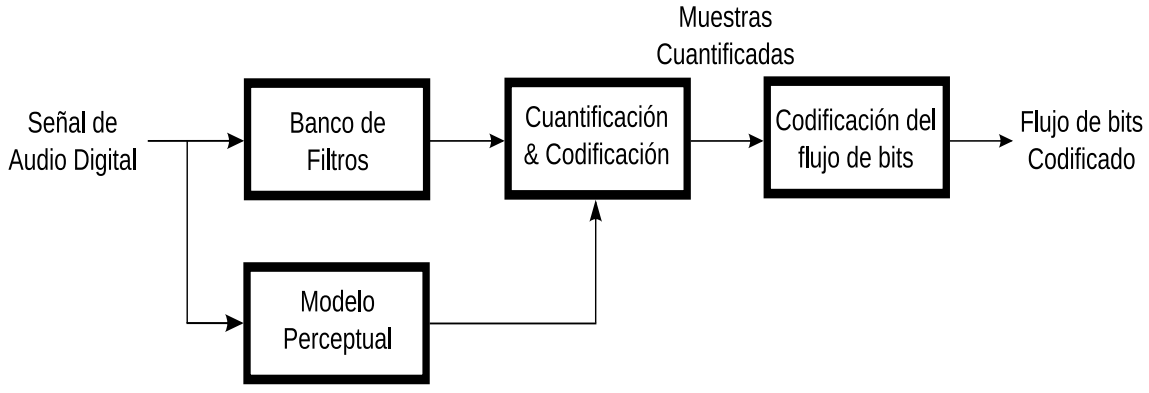

Figura 2.1: Codificador típico de MPEG-1 Layer-3.

Banco de Filtros (*Filterbank*)

El *filterbank* utilizado en MPEG-1 *Layer*-3 pertenece a los *filterbanks* híbridos. Está construido en cascada por dos tipos diferentes de *filterbanks*, el primer *filterbank* polifásico (se usa en el *Layer* I y II) y el segundo, es la Transformada Discreta del Coseno Modificada, *MDCT* por sus siglas en ingles. El *filterbank* polifásico cumple el propósito de hacer Layer-3 sea más similar a *Layer* I y II, se hace una subdivisión de cada banda de frecuencia polifásica en 18 sub-bandas más finas, aumentando la eliminación de la redundancia, lo que lleva a una mejor eficacia en la codificación de señales de tono. Otra consecuencia positiva de la resolución de frecuencia alta, la señal de error se puede controlar mejor, lo que permite un seguimiento más preciso del umbral de enmascaramiento. El *filterbank* se puede cambiar a una resolución de baja frecuencia para evitar pre-ecos.

Modelo Perceptual

El modelo perceptual o bien utiliza un *filterbank* o combina el cálculo de los valores de energía (para los cálculos de enmascaramiento) y el filterbank principal. La salida del modelo de percepción se compone de valores para el umbral de enmascaramiento o el ruido permitido para cada partición codificada. En *Layer*-III, este tipo de partición codificada es más o menos equivalente a las bandas críticas del oído humano. Si el ruido de cuantización se puede mantener por debajo del umbral de enmascaramiento para cada partición codificada, entonces el resultado de compresión debe ser indistinguible respecto a la señal original.

Cuantificación y codificación

Un sistema de dos ciclos anidados de iteración es la solución común para la cuantificación y la codificación en un codificador *Layer*-III. La cuantificación se realiza mediante un cuantizador de ley de potencia. Los valores cuantificados son codificados por la codificación de Huffman. Para adaptar el proceso de codificación a diferentes estadísticas locales de las señales de música, se selecciona de una serie de opciones de la tabla óptima de Huffman. La codificación de Huffman trabaja en pares y en el caso de números muy pequeños para ser codificados, se realiza en cuartetos. Para obtener una mejor adaptación de la señal estadística, se utilizan diferentes tablas con códigos de Huffman y así poder seleccionar una para las diferentes partes del espectro. La forma del ruido en la cuantificación se debe mantener por debajo del umbral de enmascaramiento para logarar esto se requiere de un valor de ganancia global y factores de escala que se aplican antes de cuantificación real. El proceso para hallar la ganancia óptima y los factores de escala para un bloque determinado, con tasa de bits y salida del modelo de percepción se hace generalmente por dos ciclos anidados:

- **Iteración del bucle interno (*rate loop*)**
  Las tablas de códigos Huffman asigna las palabras más cortas del código más

pequeños valores cuantificados. Si el número de bits que son resultado de la operación de codificación es superior al número de bits disponibles para codificar un bloque dado de datos, esto puede ser corregido mediante el ajuste de la ganancia global para dar lugar a un tamaño de cuantización de paso más grande, conduciendo a valores cuantificados mas pequeños. Esta operación se repite con tamaños de cuantización de paso diferentes hasta que la demanda bits resultantes de la codificación de Huffman sea lo suficientemente pequeño. El bucle se llama *rate loop*, ya que modifica el tipo de codificador general hasta que sea lo suficientemente pequeño.

- **Iteración del bucle externo (*noise control loop*)**
  Para dar forma al ruido de cuantificación de acuerdo con el umbral de enmascaramiento, los factores de escala se aplican a cada banda. Los sistemas inician con un factor de 1.0 por defecto para cada banda. Si el ruido de cuantización en una determinada banda se encuentra por encima del umbral de enmascaramiento es suplido por el modelo perceptual, el factor de escala de esta banda se ajusta para reducir el ruido de cuantificación. Dado que lograr un menor ruido de cuantificación requiere un mayor número de pasos de cuantificación y por lo tanto un mayor *bit-rate*, el ciclo de ajuste de tasa tiene que ser repetido cada vez que se utilizan nuevos factores de escala. En otras palabras, el *rate loop* está anidado dentro del *noise control loop*. El ciclo (noise control loop) exterior se ejecuta hasta que el ruido real (calculado a partir de la diferencia de los valores espectrales original menos los valores espectrales cuantificados) está por debajo del umbral de enmascaramiento.

Mientras que el ciclo de iteración interno siempre converge (si es necesario, establecer el tamaño de cuantificación de paso lo suficientemente grande como para ajustar a cero todos los valores espectrales), esto no es cierto para la combinación de ambos ciclos de iteración. Si el modelo perceptual requiere tamaños de cuantización de paso tan pequeños que el *rate loop* siempre tiene que aumentarlos para permitir la codificación en la tasa de bits requerida, ambos pueden continuar por siempre. Para evitar esta situación, varias condiciones se pueden comprobar para detener las iteraciones. Sin embargo, para una codificación rápida y con resultados de codificación buenos, tal condición debe ser evitada. Esta es una de las razones por las que un codificador MPEG *Layer*-3 por lo general necesita del modelo perceptual para cada *bit-rate*.

2.4. Estructura del formato MP3

Dentro de un archivo de audio MPEG, no hay un encabezado principal, como un archivo de audio MPEG es construido a partir de una sucesión de piezas más pequeñas llamados marcos. Cada marco es un bloque de datos con su propio encabezado y la información de audio.

En el caso de *Layer* I o II, los marcos son totalmente independientes el uno del

otro, así que usted puede cortar cualquier parte de un archivo de audio MPEG y reproducirlo correctamente. El reproductor entonces reproduce la música a partir del primer marco válida que se encuentre completo. Sin embargo, en el caso del *Layer* III, los marcos no siempre son independientes. Debido a la posible utilización de los "bytes de depósito", que es una especie de buffer interno, los marcos son a menudo dependientes el uno del otro.

Si necesita recuperar información sobre un archivo de audio MPEG, usted puede simplemente localizar el primer fotograma, y recuperar la información de su encabezado. La información dentro de otros marcos deben ser coherente con la primera, a excepción de la tasa de bits, como podría ser la recuperación de la información en un archivo de tasa de bits variable (VBR). En un archivo de VBR, la tasa de bits se puede cambiar en cada fotograma. Se puede utilizar, por ejemplo, para mantener una calidad de sonido constante durante todo el archivo, utilizando más bits cuando la música es más compleja y por tanto requiere más bits que se codificarán con una calidad similar.

El encabezado de la trama en sí es de 32 bits (4 bytes) de longitud. Los doce primeros bits (o primeros once bits en el caso de la norma MPEG 2,5) de un encabezado de la trama siempre se establecen en 1 y se llaman "sincronización de cuadros". Los marcos también pueden presentar una suma de comprobación (CRC) opcional. Esta es de 16 bits de longitud y, si existe, se encuentra inmediatamente después del encabezado. Después de la CRC vienen los datos de audio. Al volver a calcular la CRC y comparar su valor, puede comprobar si el marco ha sido alterado durante la transmisión de los bits. En las tablas 2.1, 2.2 y 2.3 se muestran los detalles del encabezado de un marco *MP3*.

Símbolo	Longitud (bits)	Descripción
A	11	Bits de sincronización
B	2	MPEG Audio versión ID 00 - MPEG Versión 2.5 (extensión de MPEG 2) 01 - reservado 10 - MPEG Versión 2 (ISO/IEC 13818-3) 11 - MPEG Versión 1 (ISO/IEC 11172-3)
C	2	*Layer* - descripción 00 - reservado 01 - *Layer* III 10 - *Layer* II 11 - *Layer* I

Cuadro 2.1: Bits del 1 al 15 del encabezado de un marco MP3.

<table>
<tr><th>Símbolo</th><th>Longitud (bits)</th><th>Descripción</th></tr>
<tr><td>D</td><td>1</td><td>Bit de Protección
0 - Protegido por CRC (CRC de 16bits después del encabezado)
1 - Sin Proteción</td></tr>
<tr><td>E</td><td>4</td><td>Índice de tasa de bits
<table>
<tr><th>bits</th><th>V1,L1</th><th>V1,L2</th><th>V1,L3</th><th>V2,L1</th><th>V2, L2 & L3</th></tr>
<tr><td>0000</td><td>free</td><td>free</td><td>free</td><td>free</td><td>free</td></tr>
<tr><td>0001</td><td>32</td><td>32</td><td>32</td><td>32</td><td>8</td></tr>
<tr><td>0010</td><td>64</td><td>48</td><td>40</td><td>48</td><td>16</td></tr>
<tr><td>0011</td><td>96</td><td>56</td><td>48</td><td>56</td><td>24</td></tr>
<tr><td>0100</td><td>128</td><td>64</td><td>56</td><td>64</td><td>32</td></tr>
<tr><td>0101</td><td>160</td><td>80</td><td>64</td><td>80</td><td>40</td></tr>
<tr><td>0110</td><td>192</td><td>96</td><td>80</td><td>96</td><td>48</td></tr>
<tr><td>0111</td><td>224</td><td>112</td><td>96</td><td>112</td><td>56</td></tr>
<tr><td>1000</td><td>256</td><td>128</td><td>112</td><td>128</td><td>64</td></tr>
<tr><td>1001</td><td>288</td><td>160</td><td>128</td><td>144</td><td>80</td></tr>
<tr><td>1010</td><td>320</td><td>192</td><td>160</td><td>160</td><td>96</td></tr>
<tr><td>1011</td><td>352</td><td>224</td><td>192</td><td>176</td><td>112</td></tr>
<tr><td>1100</td><td>384</td><td>256</td><td>224</td><td>192</td><td>128</td></tr>
<tr><td>1101</td><td>416</td><td>320</td><td>256</td><td>224</td><td>144</td></tr>
<tr><td>1110</td><td>448</td><td>384</td><td>320</td><td>256</td><td>160</td></tr>
<tr><td>1111</td><td>bad</td><td>bad</td><td>bad</td><td>bad</td><td>bad</td></tr>
</table>
NOTA: Todos los valores están en kbps
V1 - MPEG Versión 1
V2 - MPEG Versión 2 and Versión 2.5
L1 - Layer I
L2 - Layer II
L3 - Layer III</td></tr>
<tr><td>F</td><td>2</td><td>Índice de frecuencias de muestreo
<table>
<tr><th>bits</th><th>MPEG1</th><th>MPEG2</th><th>MPEG2.5</th></tr>
<tr><td>00</td><td>44100 Hz</td><td>22050 Hz</td><td>11025 Hz</td></tr>
<tr><td>01</td><td>48000 Hz</td><td>24000 Hz</td><td>12000 Hz</td></tr>
<tr><td>10</td><td>32000 Hz</td><td>16000 Hz</td><td>8000 Hz</td></tr>
<tr><td>11</td><td>reservado</td><td>reservado</td><td>reservado</td></tr>
</table></td></tr>
<tr><td>G</td><td>1</td><td>Bit de relleno
0 - El marco no es rrellenado
1 - El marco es rellenado con una ranura extra</td></tr>
</table>

Cuadro 2.2: Bits del 16 al 23 del encabezado de un marco MP3.

<table>
<tr><th>Símbolo</th><th>Longitud (bits)</th><th>Descripción</th></tr>
<tr><td>H</td><td>1</td><td>Bit Privado. Este es solo informativo</td></tr>
<tr><td>I</td><td>2</td><td>Modo (Canales)
00 - Estéreo
01 - Joint Stereo (Estéreo)
10 - Doble Canal (2 canales Mono)
11 - Un Canal (Mono)</td></tr>
<tr><td>J</td><td>2</td><td>Modo extendido (Sólo se utiliza con Joint Stereo)
<table>
<tr><th></th><th>Layer I and I</th><th colspan="2">Layer III</th></tr>
<tr><td>valor</td><td>Layer I & II</td><td>Intensidad estéreo</td><td>MS stereo</td></tr>
<tr><td>00</td><td>bandas 4 to 31</td><td>apagada</td><td>apagada</td></tr>
<tr><td>01</td><td>bandas 8 to 31</td><td>encendida</td><td>apagada</td></tr>
<tr><td>10</td><td>bandas 12 to 31</td><td>apagada</td><td>encendida</td></tr>
<tr><td>11</td><td>bandas 16 to 31</td><td>encendida</td><td>encendida</td></tr>
</table></td></tr>
<tr><td>K</td><td>1</td><td>Copyright
0 - Audio sin copyright
1 - Audio con copyright</td></tr>
<tr><td>L</td><td>1</td><td>Original
0 - Copia del medio original
1 - Medio Original</td></tr>
<tr><td>M</td><td>2</td><td>Énfasis
00 - ninguno
01 - 50/15 ms
10 - reservado
11 - CCIT J.17</td></tr>
</table>

Cuadro 2.3: Bits del 24 al 32 del encabezado de un marco MP3.

2.5. Software LAME

2.5.1. LAME

LAME[7] es un codificador de alta calidad de MPEG Audio Layer III (MP3) licenciado bajo la LGPL[23].

El desarrollo de LAME comenzó a mediados de 1998. Mike Cheng comenzó como un parche de las fuentes del codificador MP3-8Hz. Después de algunos problemas de calidad planteadas por los demás, decidió empezar de cero basado en las fuentes dist10. Su objetivo era sólo acelerar las fuentes de ISO(dist10)[16], y dejar intacta su calidad. Esa rama (un parche contra las fuentes de referencia) se convirtió en Lame 2.0, y sólo en Lame 3.81 se sustituye todo el código de dist10, haciendo LAME más que sólo un parche.

El proyecto se convirtió rápidamente en un proyecto de equipo. Mike Cheng abandonó el liderazgo y comenzó a trabajar en tooLame[36][7], un codificador de MP2. Mark Taylor se convirtió en líder y comenzó a perseguir una mayor calidad, además de una mayor velocidad. Él puede ser considerado como el iniciador del proyecto LAME en su forma actual. Se publicó la versión 3.0 con *gpsycho*, un nuevo modelo psicoacústico que él desarrolló. A principios de 2003 Mark dejó el liderazgo del proyecto, y desde entonces el proyecto ha sido liderado por la cooperación activa de los desarrolladores.

Hoy en día, LAME es considerado el mejor codificador *MP3* con altas y bajas tasas de bits y tasas de bits variables (VBR), sobre todo gracias al trabajo dedicado de sus desarrolladores y el modelo de licenciamiento de software libre permitió que el proyecto aproveche los recursos de ingeniería de todo el mundo. Tanto las mejoras calidad y velocidad siguen pasando, seguramente haciendo a LAME el único codificador *MP3* que se encuentra en desarrollo activo.

LAME esta en constante actualización, atento a las actualizaciones de los estándares establecidos por la ISO[16], estas actualizaciones hacen que la codificación *MP3* obtenga una mayor calidad.

El algoritmo general de LAME se muestra en la Figura 2.2

2.5.2. Archivos fuentes

A continuación se describen los archivos más importantes de LAME *encoder* los cuales se muestran el la Figura 2.3.

encoder.c
Contiene las funciones para la codificación, en caso de que sea *join stereo* los canales

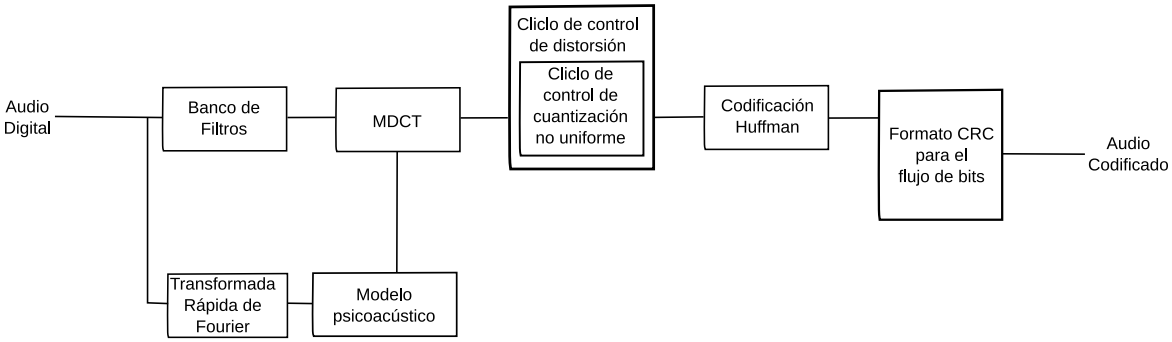

Figura 2.2: Lame Codificador de MPEG Layer-3.

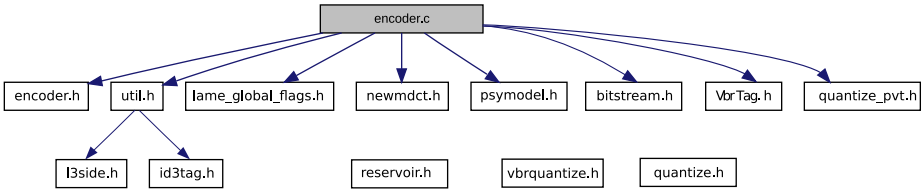

Figura 2.3: Archivos fuentes de Lame.

se separan para una codificación separada devolviendo un solo marco *MP3*.

encoder.h
Implementación de la función para la codificación así como la definición de contantes para la misma.

util.h
Se define una estructura que contiene los distintos parametros para codificar un archivo a *MP3*, así como definiciones y funciones útiles para los calculos que esta requiere.

lame_global_flags.h
Se define la estructura global del programa que incluye las variables para el manejo de parametros para la codificación.

l3side.h
En este se definen una serie de estructuras para el manejo del *Layer* 3.

id3tag.h
Se definen una seria de estructuras que corresponden a la identificación del tipo de musica, autor, genero etc.

newmdct.h
Implementación de la Trasformada Discreta del Coseno Modificada (MDCT).

psymodel.h
Implementación de modelo psicoacústico.

quantize.h y quantize_pvt.h
Implementación de la cuantización *MP3*.

bitstream.h
Interfaz de salida de flujo de bits de *MP3*, actualización del CRC y codificación de Huffman.

VbrTag.h
Se define un encabezado para el archivo *MP3*, el cual tiene el número de muestras que contenidas en un *bitstream*, el número de bytes contenidos en un bitstream y una tabla de contenido para el acceso aleatorio.

vbrquantize.h
Implementación de la cuantización con bitrate variable.

reservoir.h
Implementación de funciones para analizar las muestras y en caso de que estén incompletas o al codificarlas pierden calidad se le aplican los bytes de depósito.

2.5.3. Software que utiliza LAME

Software Libre que utiliza LAME:

- FFmpeg[6]: utiliza LAME para la codificación mp3
- Audacity[24]: Editor de Audio, utiliza LAME para la exportación a mp3
- k3b[25]: utiliza LAME para convertir CD's de audio a mp3
- Arson[26]: utiliza LAME para convertir CD's de audio a mp3
- iTunes-LAME[27]: codificador mp3 para MacOS el cual utiliza LAME
- SecondSpin[28]: utiliza LAME para la codificación mp3

Software Comercial que utiliza LAME:

- WINAMP[29]: utiliza LAME para convertir CD's de audio a mp3
- Tracktion3[30]: utiliza LAME para la exportación a mp3
- Acoustica[31]: utiliza LAME para la exportación a mp3
- Audion[32]: utiliza LAME para la codificación mp3
- CD Copy[33]: utiliza LAME para convertir CD's de audio a mp3

Capítulo 3

Desarrollo de la Aplicación

3.1. Metodología y Codificador a Paralelizar

Analizando el algoritmo de codificación se llegó a la conclución de que es altamente paralelizable debido a que no existe dependencia de datos, es decir la codificación de muestras de audio no depende de resultados anteriores con lo cual el algoritmo se puede dividir en cuatro tareas: el reconocimiento del archivo, la división del archivo de audio, la codificación de cada parte y la unión de las partes codificadas, para lo cual se diseñó e implementó un algoritmo para cada tarea y su interacción se muestra en la Figura 3.1, en la cual destaca el bloque de procesamineto paralelo que contiene a los hilos de codificación los cuales se encuentran dentro del proceso de codificación. El número de hilos dependera de la arquitectura.

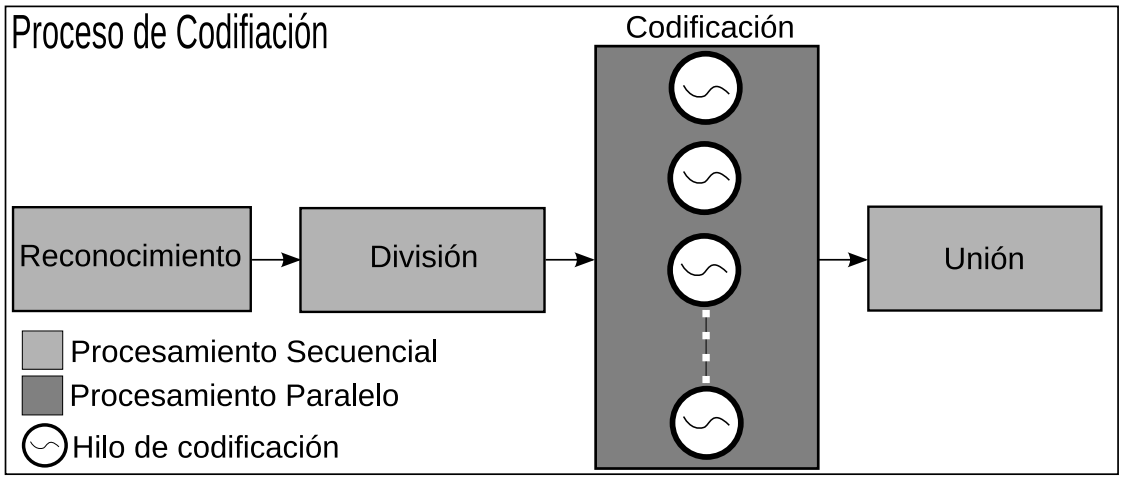

Figura 3.1: Codificación concurrente.

Se paralelizará el software LAME 3.98.2[7] el cual se describe en el Capítulo 2. Se optó por dicho software por las siguientes razones:

- Software de código abierto
- Documentación
- Su desarrollo se basa en los estándares de codificación *MP3*
- Se encuentra en constante desarrollo y actualización
- Tanto software libre como comercial utilizan LAME como codificador *MP3*

Especificaciones de la Codificación:

- CBR (constant bitrate)
- bitarate de 128Kbps
- samplerate de 44.1 KHz
- *joint-stereo*

3.2. Programación Multi-Hilo

Técnicamente un hilo es definido como un flujo de instrucciones independientes que puede ser planificadas para su ejecución por el sistema operativo. Si un programa contiene una serie de procedimientos, después estos procedimientos pueden ser programados de tal forma que su ejecución sea simultánea y/o de forma independiente obteniendo así una ejecución multi-hilo (multi-threaded). Otra forma de ver un hilo es como un subproceso o proceso ligero esto debido a que solo contiene las caracteristicas esenciales para poder ser ejecutado de forma independiente en la Figura 3.2 muestra la diferencia entre un hilo y un proceso.

La motivación principal para usar programación multi-hilo es encontrar mejoras en el rendimiento del programa. Cuando se compara el costo de crear y gestionar un proceso, un hilo se puede crear con una carga mucho menor para el sistema operativo. La gestión de hilos requiere menos recursos del sistema que la gestión de procesos. En la actualidad con las arquitecturas multi-núcleo, los hilos son ideales para la programación paralela pero existen consideraciones para el diseño de programas paralelos, tales como: particionamiento, balanceo de carga, sincronización, condiciones de competencia y complejidad de un programa.

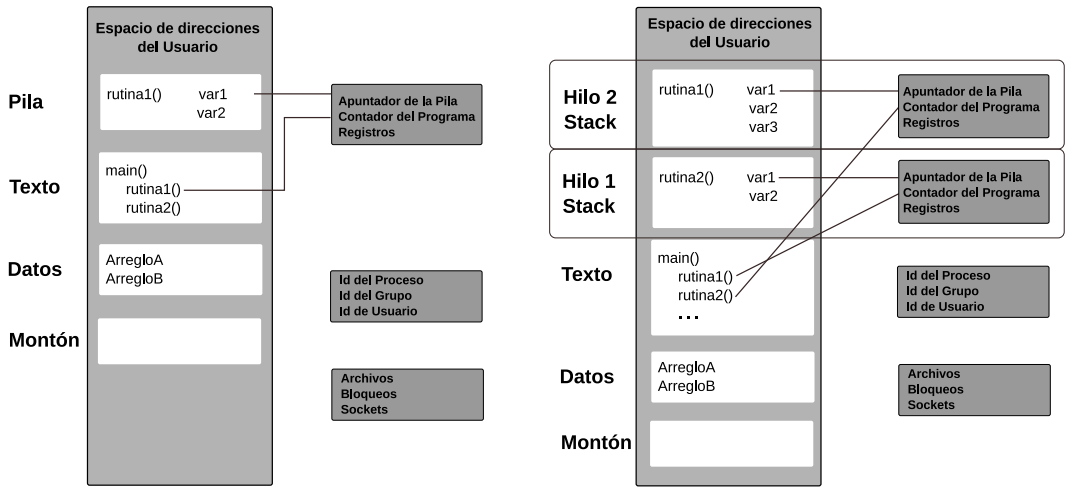

Figura 3.2: Proceso y Hilos dentro de un proceso.

3.3. Módulo de Reconocimiento

El módulo permite determinar si el archivo a codificar es un archivo *WAV*, analizando el encabezado del archivo,. Si el archivo examinado resulta ser *WAV*, se guardará la información del encabezado de lo contrario el programa terminará.

Formato de Audio *WAV*: *WAV* [12][13], apócope de *WAVEform audio format*, es un formato de audio digital sin compresión de datos. Al ser un archivo sin compresión es muy grande en tamaño, este formato contiene información que no es perceptible al oído humano. Este tipo de archivo de audio es muy utilizado por profesionales. El formato *WAV* es una variante del formato *RIFF* (Resource Interchange File Format, formato de archivo para intercambio de recursos). La estructura del formato *RIFF* se muestra en la Figura 3.3.

Figura 3.3: Estructura del formato *RIFF*.

Descripción de los miembros del formato *RIFF* para que sea un archivo WAV válido.

- RIFF Id: Contiene los caracteres “RIFF” (0x52494646).
- Tamaño RIFF: Contiene el tamaño total del archivo.
- Formato: Contiene los caracteres “WAVE” (0x57415645).
- fmt Id: Contiene los caracteres “fmt ” (0x666d7420).
- Tamaño fmt: Contiene el tamaño del bloque ”fmt ” (16 bytes).
- Formato de Audio: Contiene el formato de las muestras PCM=1 (0x0001).
- Numero de Canales: Mono = 1, Stereo = 2.
- Frecuencia de muestreo(Hz) = 8000, 44100, .. etc.
- Tranferencia de bytes por segundo: Frecuencia de Muestreo * Numero de Canales * (Bits por muestra/8).
- Tamaño de un bloque=Numero de Canales * (Bits por muestra/8).
- Bits por muestra: número de bits que forman una muestra.
- Id Datos: Contiene los caracteres “data” (0x64617461).
- Tamaño Datos: el tamaño del bloque de datos:
- Datos: Contiene las muestras del archivo wav.

Diseño del Módulo de Reconocimiento

En base a la información recopilada sobre el formato *WAV* se diseñó el Módulo de Reconocimiento. El cual se basa en la lectura campo a campo del encabezado del archivo, tomando en cuenta el orden[1] de los datos (little-endian o big-endian), para lo cual se implementaron funciones para la lectura en de datos en orden big-endian y trasformarlos a little-endian.

- Las funciones de lectura se pueden observar en la Figura 3.4 y 3.5.
- La función de Reconocimiento se muestra en la Figura 3.6

[1]Endian: Designa el formato en el que se almacenan los datos de más de un byte en un ordenador. Big-endian representa los datos de forma tal que el byte más signicativo en la dirección más baja, mientras que Little-endian representa los datos de forma tal que el byte menos signicativo en la dirección más baja

```
Result: Regresa 2 bytes en orden little-endian
Read16BitsHighLow(File) begin
    int first, second, result;
    first := 0xff & getc(File);
    second := 0xff & getc(File);
    result := (first << 8) + second;
    regresa result;
end
```

Figura 3.4: Funciones para la lectura de 2 bytes Big-endian y trasformarlos a Little-endian

```
Result: Regresa 4 bytes en orden little-endian
Read32BitsHighLow(File) begin
    int first, second, result;
    first := 0xffff & Read16BitsHighLow(File);
    second := 0xffff & Read16BitsHighLow(File);
    result := (first << 16) + second;
    return (result);
end
```

Figura 3.5: Funciones para la lectura de 4 bytes Big-endian y trasformarlos a Little-endian

```
Data: La entrada es un Archivo (File)
Result: Si el archivo entrante es WAV regresa Verdadero en otro caso
        regresara Falso
ParseHeader(File) begin
    RIFFId := Read32BitsHighLow(File);
    if RIFFId="RIFF" then
        fileSize := Read32BitsHighLow(File);
        Format := Read32BitsHighLow(File);
        if Format="WAVE" then
            FmtId = Read32BitsHighLow(File);
            if FmtId="fmt " then
                FmtSize=:Read32BitsLowHigh(File);
                AudioFormat := Read16BitsLowHigh(File);
                if AudioFormat=1 then
                    Canales := Read16BitsLowHigh(File);
                    Samplerate := Read32BitsLowHigh(File);
                    ByteRate := Read32BitsLowHigh(File);
                    BlockAlign := Read16BitsLowHigh(File);
                    bitsPerSample := Read16BitsLowHigh(File);
                    DataID := Read32BitsHighLow(File);
                    if DataId="data" then
                        DataSize := Read32BitsLowHigh(File);
                    else
                        regresa falso;
                    end
                else
                    regresa falso;
                end
            else
                regresa falso;
            end
        else
            regresa falso;
        end
    else
        regresa falso;
    end
    regresa verdadero;
end
```

Figura 3.6: Función para el reconocimiento de archivos *WAV*

3.4. Módulo de División

Una vez que el archivo fue verificado este módulo verifica cuántos núcleos contiene la arquitectura, a partir de esto se calcula la posición desde la cual cada hilo leera del archivo *WAV*, así como la posición de fin de lectura. Esto se hace a través del identificador de cada hilo el cual es un número consecutivo que representa el número de partición, tambien se genera un nombre de archivo temporal en el cual se guardará el audio codificado por cada hilo, el funcionamiento de este módulo se muestra en la Figura 3.7. La información de este módulo se guardará en un arreglo de estructuras tipo **thread_encoder** la cual se muestra en la Figura 3.8.

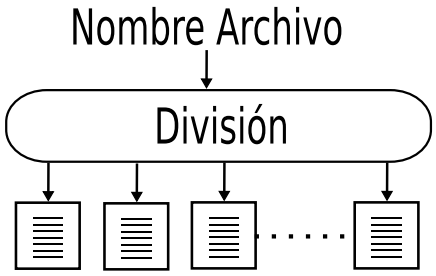

Figura 3.7: Módulo de división.

```
/*** Estructura que se envia como argumento a cada hilo ***/
typedef struct thread_encoder_struct
begin
    int id;
    char *inPath;
    char *outPath;
    int start;
    int end;
    mp3_flags *gof;
end
thread_encoder;
```

Figura 3.8: Estructura que se enviara como argumento a cada hilo.

En la estructura se guarda el id , el nombre del arhichivo *WAV*, un nombre temporal distinto para cada hilo, posición de inicio de lectura en bytes y posición de fin de lectura en bytes y los parámetros de la codificación.

Número de núcleos

Para determinar el número de núcleos que tiene la arquitectura, GNU/Linux cuenta con el archivo */proc/cpuinfo*, el cual contiene información sobre el o los CPU(s) de la computadora. El archivo tiene la siguiente estructura por cada núcleo:

```
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model    : 15
model name  : Intel(R) Core(TM)2 Duo CPU     T5670  \@ 1.80GHz
stepping : 13
cpu MHz     : 800.000
cache size  : 2048 KB
physical id : 0
siblings : 2
core id : 1
cpu cores : 2
.
.
.
```

Para saber cuantos núcleos tiene la arquitectura se cuenta el número de veces que aparece la palabra **processor** en el archivo , la función se muestra en la Figura 3.9.

```
Result: Regresa el numero de núcleos
substr(buffer,inicio,longitud)
int get_num_cores()() begin
    buffer;
    cores:=0;
    cpufile:=fopen("/proc/cpuinfo");
    buffer:=getline(cpufile);
    while buffer ≠ NULL do
        if substr(buffer,0,9) = "processor" then
            cores++;
        end
        buffer:=getline(cpufile);
    end
    return cores;
end
```

Figura 3.9: Función para determinar el número de núcleos de la arquitectura.

Posiciones de lectura

Para el cálculo de la posición de lectura se requiere de tres datos los cuales son: el número total de particiones, el tamaño del bloque de datos del archivo *WAV*, el identificador de la partición. Apartir de estos datos se calcula la posición de inicio y fin de lectura para cada hilo, la función se muestra en la Figura 3.10.

```
partition(id,format,n,*start,*end)  begin
    framesize:=1152*fromat->Channels*(format->Bits_Per_Sample/8);
    offset:=format->Data_Size/(framesize*partition);
    *start:=( id*offset*framesize)+44;
    if id!=(partition-1) then
        *end:=( (id+1)*offset*framesize)+44;
    else
        *end:=format->Data_Size+44;
    end
end
```

Figura 3.10: Función para determinar la posición de inicio y fin de lectura para cada hilo.

Nombres Temporales

La generación de los nombres para los archivos temporales se hace a partir del nombre del archivo de entrada y el id de la partición, la función que genera los nombres se muestra en la Figura 3.11.

```
/***Función que genera un nombre de archivo temporal***/
char * init_outfile(char * inPath,char * outPath, int id)  begin
    int len;
    len=strlen(inPath);
    outPath=substr(inPath,0,len-3);
    outPath=strncat(outPath,"mp3");
    outPath[len]=(id/100)+48;
    outPath[len+1]=((id%100)/10)+48;
    outPath[len+2]=((id%100)%10)+48;
    outPath[len+3]=0;
    return outPath;
end
```

Figura 3.11: Función que genera un nombre de archivo temporal para cada hilo.

3.5. Módulo de Codificación

Para que la codificación se lleve acabo de forma concurrente se hará uso de la programación multi-hilo mediante la biblioteca **pthreads**[21] de POSIX[34]. Se modificó la función de codificación de LAME para convertirla en un hilo, con lo cual la codificación de cada hilo creado sera independiente.

La función de codificación de LAME realiza un procesamiento secuencial el cual leia muestra por muestra de audio para su posterior codificación como se muestra en la Figura 3.12, lo cual hace que el tiempo de ejecución dependa unicamente del tamaño del archivo de entrada.

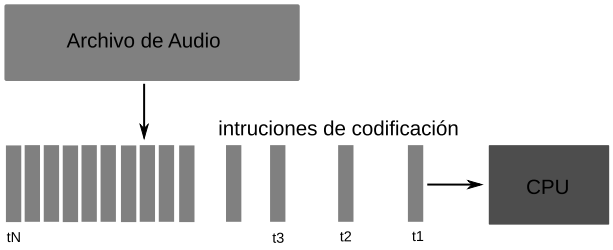

Figura 3.12: Codificación Secuencial.

Con la programación multi-hilo se logró hacer que cada hilo leea una determinada cantidad de muestras de audio y codificarlas concurrentemente como se muestra en la Figura 3.13, haciendo que el tiempo de ejecución no solo dependa del tamaño del archivo sino también de la arquitectura en la que se este ejecutando, obteniendo así un mayor rendimiento ya que de esta forma se hace un mejor uso de los recursos del sistema. El audio codificado por cada hilo se almacenará en archivos temporales, esto para evitar condiciones de competencia ya que estas podrian afectar el rendimiento de la aplicación, dichos archivos serán unidos al finalizar la codificación.

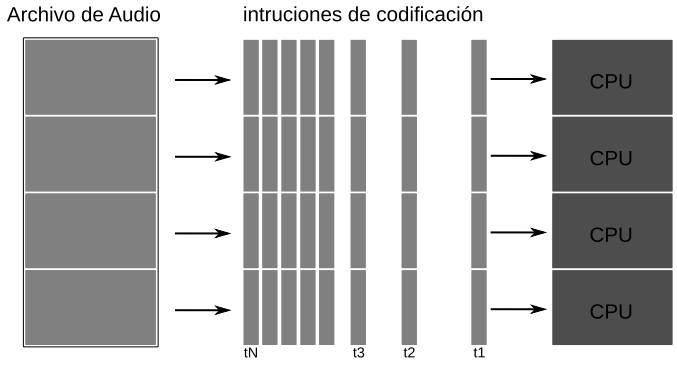

Figura 3.13: Codificación Concurrente.

Las modificaciones que se le hicieron a la función de codificación de LAME son las siguientes:

- Cambio en la definición de la función ya que para la creación de un hilo se requiere un apuntador a una función, dicha función debe retornar void y tiene un único parametro el cual es un apuntador a void
- La condición de paro de lectura cambio ahora en vez de esperar el fin de archivo, leera una determinada cantidad de bytes
- Cada hilo creara un archivo temporal para que el siguiente módulo los una

En la Figura 3.14 se muestran las modificaciones realizadas.

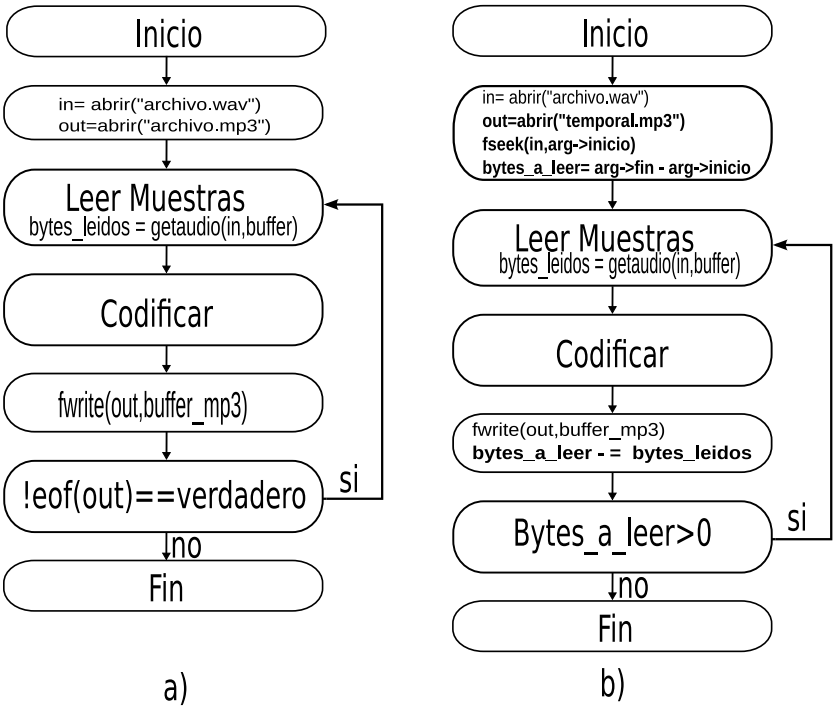

Figura 3.14: a) Codificación de LAME. b) Hilo de codificación.

A continuación se muestra la definición de la nueva función y llamada a la función **pthread_create**[21], utilizando la nueva función, donde se le envían como parámetros una referencia para el indentificador del hilo, los atributos del hilo, la referencia de la función de codificación y una estructura tipo **thread_encoder**.

```
/***Defición de función y llamada a pthread_create***/
void *thread_function_encoder(void *args);
pthread_create(&thread_id,NULL,thread_function_encoder,&thread_arg);
```

3.6. Módulo de Unión

El módulo se encargará de la unión de los distintos archivos generados por el módulo anterior, el módulo recibirá un arreglo de estructuras tipo **thread_encoder**, como se describió anteriormente en la estructura se encuentra un id el cual representa el número de partición y el nombre del archivo temporal. A partir de estas estructuras se abrirá uno a uno los archivos y se copiarán de forma ordenada respecto al id, en un archivo nuevo el cual sera el archivo final *MP3*, este archivo contendra el audio codificado. Cada vez que se termina de leer un archivo temporal este se elimina. El funcionamineto del módulo se muestra en la Figura 3.15 y el código del mismo se muestra en la Figura 3.16. El buffer tiene el tamaño de una marco MP3 para asegurar que no se instrodusca ruido en el proceso.

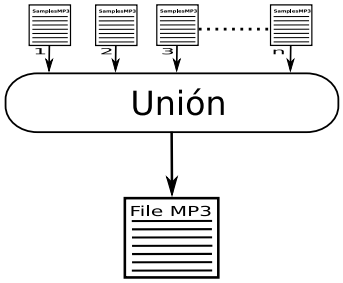

Figura 3.15: Módulo de unión.

```
void union_mp3(thread_file *tf,thread_encoder *te)  begin
    buffer[MAXMP3BUFFER];
    FILE outfile,infile;
    outfile=open(tf->outPath,"w+b");
    for i=0;i<tf->partition;i++ do
        infile=open(te[i].outPath,rb");
        iread=fread(buffer,MAXMP3BUFFER,infile);
        while iread>0 do
            fwrite(buffer,sizeof(unsigned char),iread,outfile);
            iread=fread(buffer,sizeof(unsigned char),MAXMP3BUFFER,infile);
        end
        close(infile);
        elimina(te[i].outPath);
    end
    close(outfile);
end
```

Figura 3.16: Función que realiza la unión y eliminación de los archivos temporales.

3.7. Interfaz Gráfica de Usuario

En esta sección se mostrra el desarrollo de la GUI para el software desarrollado. La GUI se desarrollo con el framework de Qt[5]. La interfaz esta basada en una ventana principal, la cual interactua con los componentes que forman parte de la misma, dichos componentes puden ser marcos, menus, barras de herramientas, barra de estado entre otros. La disposición tiene un área central que pueda ser ocupada por cualquier tipo de componente. Se muestra la disposición general de la GUI en la Figura 3.17.

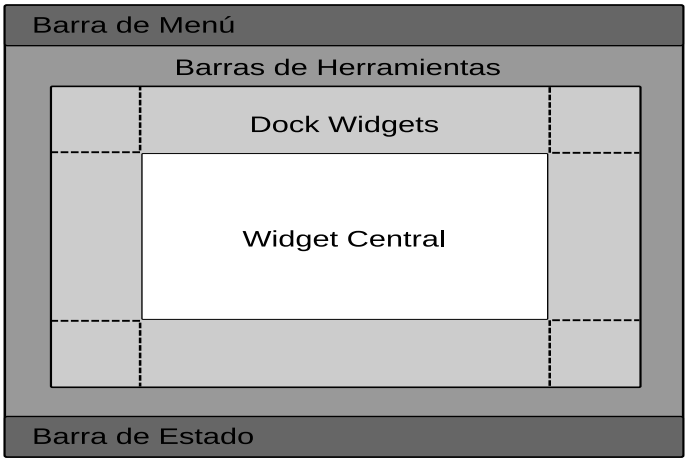

Figura 3.17: Disposición de una Ventana Principal

La GUI permitira añadir mas de un archivo para su codificación, elegir el número de partes en las que se codificara cada archivo así como poder detener la codificación en cualquier momento. El diseño de la GUI contendra los siguientes componentes:

- Una tabla en la cual se mostrara la infomación de los archivos a codificar y la opción de elegir el número de partes en las que se particionara en archivo
- Una barra de herramientas con las siguientes opciones:
  - Añadir archivo
  - Remover archivo
  - Limpiar tabla
  - Iniciar la codificación
  - Detener la codificación
- Un marco con las especificaciones de la codificación
- Un marco con información del equipo así como el número de núcleos que contiene
- Un marco en el cúal se prodra elegir la ruta destino de los archivos codificados

3.7.1. Diseño de la GUI

El la Figura 3.18 se muestra la GUI desarrollada.

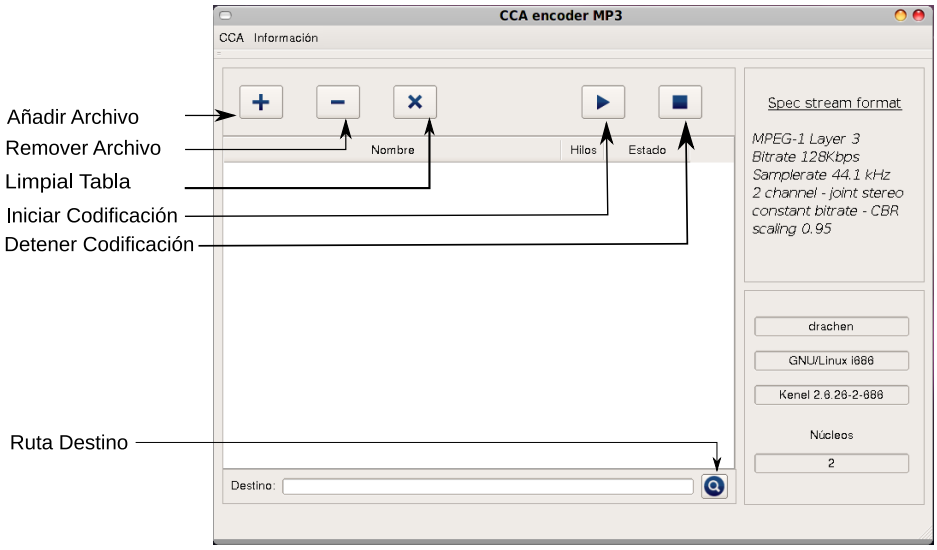

Figura 3.18: GUI

3.7.2. Casos de Uso

El objetivo de los casos de uso es captar los requerimientos de los usuarios. Consiste en la interacción de los actores principales con el software.

Actores Principales
Se identifico un solo actor el cual es el **Usuario**.

En la Figura 3.19 se muestran los casos de uso.

Añadir archivo

Descripción En este el usuario prodra añadir un archivo para su posterior codificaión.
Pre-Condiciones El usuario debe elegir la ruta destino primero.
Flujo Principal Se muestra la Ventana Principal y el usuario eligio de la barra de herraminetas la opción de Añadir Archivo.
Sub-Flujos Se mostrara un cuadro de dialogo tipo explorador en el cual el usuario podra elegir un archivo,este se enviara al módulo de reconocimiento para ser validado como archivo de audio *WAV* y se agrega a la tabla.
Exepciones En caso de que el archivo elegido no sea *WAV* se mostra un mensaje y el archivo no sera añadido.

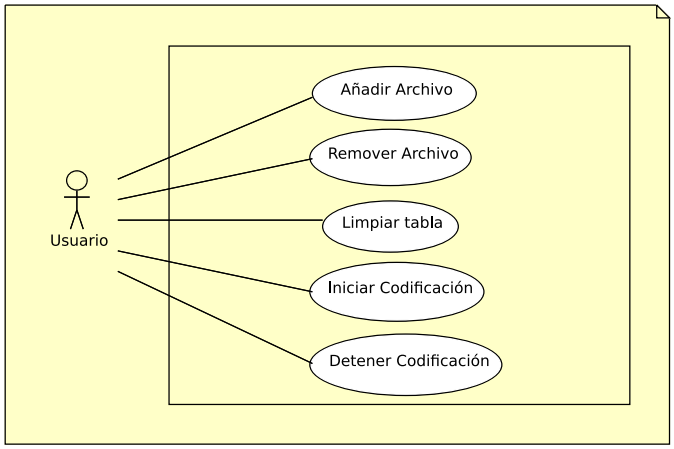

Figura 3.19: Diagrama de Casos de Uso del Usuario

Remover archivo

Descripción En este el usuario prodra remover un archivo previamente añadido a codificar en la tabla de archivos.
Pre-Condiciones El usuario debe selecionar una fila de la tabla.
Flujo Principal Se muestra la Ventana Principal y el usuario eligio de la barra de herraminetas la opción de Remover Archivo.
Sub-Flujos Se removera el archivo elegido de la tabla
Exepciones En caso de que la tabla este vacia o no se selecciono un archivo la acción sera nula.

Limpar Tabla

Descripción En este el usuario prodra remover todo loa archios añadidos.
Flujo Principal Se muestra la Ventana Principal y el usuario eligio de la barra de herraminetas la opción de Limpiar Tabla.
Sub-Flujos Se removeran todos los archivos de la tabla
Exepciones En caso de que la tabla este vacia la acción sera nula.

Ruta destino

Descripción En este el usuario prodra elejir la ruta en la cual se guardaran los archivos *MP3*.
Flujo Principal Se muestra la Ventana Principal y el usuario eligio del frame inferior Ruta Destino
Sub-Flujos Se removeran todos los archivos de la tabla

Iniciar Codificación

Descripción En este el usuario inicia el proceso de codificación de los archivos selecionados.
Pre-Condiciones El usuario debe selecionar una fila de la tabla.
Flujo Principal Se muestra la Ventana Principal y el usuario eligio de la barra de herramientas Iniciar Codificación
Sub-Flujos Se codificaran los archivos añadidos y se guardaran los archivos codificados en la ruta destino
Exepciones En caso de que la tabla de archivos este vacia se mostrara un mesaje.

Detener Codificación

Descripción En este el usuario dendra el proceso de codificación.
Pre-Condiciones La codificación debe haber iniciado.
Flujo Principal Se muestra la Ventana Principal y el usuario eligio de la barra de herramientas Detener Codificación
Sub-Flujos Detendra la codificación
Exepciones En caso de que la codificación no este iniciada se mostrara un mesaje.

3.7.3. Acerca de ..

También se desarrollo una interface donde se muestran los datos del sofware como son:

- Nombre del Software
- Descripción
- Versión
- Autor
- Contacto
- Sitio donde se puede descragar el software
- Licencia

Esta interface se muestra en la Figura 3.20 y es accecible desde el menú **Información->Acerca de ..**, de la ventana principal.

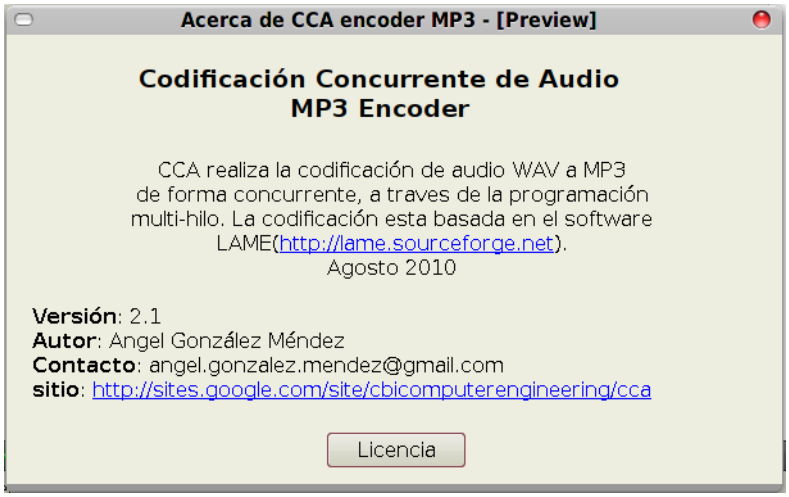

Figura 3.20: Interface Acerca de ..

Capítulo 4

Resultados

En este capítulo se presentan los resultados obtenidos de la codificación de audio por medio del software descrito en el Capítulo 3.

Al aplicar la programación multi-hilo sobre una solución secuencial de codificación, se obtuvo como resultado el software con la estructura mostrada en la Figura 3.1. La característica mas notable del software desarrollado es que aprovecha los recursos de la arquitectura en la cual se esta ejecutando.

4.1. Plataformas de Prueba

El software se probo en dos computadoras, las caracteristicas de dichas computadoras se muestran a continuación:

Equipos1-Laptop

- **S.O.** GNU/Linux Debian 5.0.5, 32bits
- **Procesador** Intel(R) Core(TM)2 Duo CPU T5670 1.80GHz
- **Numero de Núcleos** 2 a 800MHz cada uno
- **Memoria Cache L2** 2MB de chace L2
- **Memoria RAM** 3 GB

Equipo2-Servidor

- **S.O.** GNU/Linux Debian squeeze, 64bits
- **Procesador** Intel(R) Xeon(R) CPU E5540 2.53GHz

- **Numero de Núcleos** 16 a 2.53GHz cada uno
- **Memoria Cache L2** 8MB
- **Memoria RAM** 31.5 GB

4.2. Verificación del Software

La primera prueba a realizar es verificar el correcto funcionamiento del software desarrollado. Para ello se realizaron experimentos los cuales consistian en la codificación de audio *WAV* a *MP3* utilizando el software desarrollado y posteriormente se verifico que el audio codificado no contubiera ruido, esto se hizo a través de *mplayer*[35] el cual es un reproductor de audio, dicho software muestra mensajes en caso de que una marco *MP3* este corrupto o incompleto.

Al finalizar los experimentos también se comparo que la duración de reproducción del archivo codificado tanto con el software desarrollodo y LAME sea la misma, en ambos casos el resultado fue el mismo con lo cual se garantiza que el proceso de codificación es correcto.

4.3. Tiempo de Codificación

Comprobado el correcto funcionamiento del sofware dasarrollado, se procedió a realizar experimentos para medir el tiempo de codificación. Los experimentos consistieron en medir el tiempo codificación en las distintas plataformas tanto para LAME, como para una software desarrollado. Con ambos resultados es posible evaluar la aceleración obtenida al utilizar la programación multi-hilo sobre la programación secuencial.

La medición del tiempo de codificación del sofware desarrollado se llevo a cabo con la llamada la función ***gettimeofday()***, definida en ***/usr/include/sys/time.h***. La función ***gettimeofday()*** regresa en una estructura con el número de segundos transcurridos a partir de las **00:00:00 horas del 1 de enero de 1970**. Para calcular en tiempo de codificación se manda a llamar a dicha función al inicio y al final de la codificación, a partir de los valores devueltos se hace una diferencia entre el tiempo inicial y el final, con lo cual se obtiene el tiempo de codificación.

4.4. Archivos

Para las pruebas se eligieron 3 archivos de audio *WAV* los cuales se muestran en la Tabla 4.1.

Archivos WAV		
Nombre	**Tamaño(MB)**	**Frames**
Audio1	42	9419
Audio2	133	30236
Audio3	531	120719

Cuadro 4.1: Archivos de audio *WAV*.

4.5. Codificación con LAME

A continuación se mostraran los resultados de codificar los archivos descritos en la Tabla 4.1 con LAME en los equipos mencionados en la Sección 4.1.

Al codificar los archivos, se obtuvieron los tiempos de codificación mostrados en la Tabla 4.2.

	Tiempo de Codificación		
Equipo	**Archivo1** [seg]	**Archivo2** [seg]	**Archivo3** [seg]
Equipo1	18.475	58.94	231.259
Equipo2	10.708	34.566	135.069

Cuadro 4.2: Tiempos de Codificación con LAME.

4.6. Codificación con el software desarrollado

A continuación se mostraran los resultados de codificar los archivos descritos en la Tabla 4.1 con el software desarrollado en los equipos mencionados en la Sección 4.1.

Al codificar los archivos, se obtuvieron los tiempos de ejecución mostrados las Tablas 4.3 y 4.4.

	Tiempo de Codificación		
Hilos	**Archivo1** [seg]	**Archivo2** [seg]	**Archivo3** [seg]
1	17.115	55.333	218.386
2	9.565	30.826	122.394

Cuadro 4.3: Tiempos de Codificación en Equipo1 con el software desarrollado.

	Tiempo de Codificación		
Hilos	**Archivo1** [seg]	**Archivo2** [seg]	**Archivo3** [seg]
1	10.342	33.953	134.639
2	5.517	17.824	70.683
3	3.702	12.098	47.532
4	2.803	9.035	35.553
5	2.262	7.249	28.549
6	1.903	6.136	23.988
7	1.627	5.274	20.517
8	1.457	4.601	17.864
9	1.685	5.371	19.817
10	1.509	4.881	18.745
11	1.418	4.600	17.887
12	1.308	4.226	16.406
13	1.286	4.091	15.845
14	1.226	3.894	14.949
15	1.204	3.718	14.409
16	1.114	3.557	13.843

Cuadro 4.4: Tiempos de Codificación en Equipo2 con el software desarrollado.

4.7. Aceleración

En base a los resultados obtenidos en las secciones 4.5 y 4.6, se cálculo la aceleración por medio de la ecuación 4.1. En las Tablas 4.5 y 4.6 se muestra la aceleración obtenida en cada caso.

$$A = \frac{Tiempo_{secuencial}}{Tiempo_{concurrente}} \tag{4.1}$$

	Aceleración		
Hilos	**Archivo1**	**Archivo2**	**Archivo3**
1	1.07946	1.06519	1.05895
2	1.93152	1.91202	1.88946

Cuadro 4.5: Aceleración obtenida en el Equipo1.

	Aceleración		
Hilos	**Archivo1**	**Archivo2**	**Archivo3**
1	1.03539	1.01805	1.00319
2	1.94091	1.9393	1.91091
3	2.89249	2.85717	2.84164
4	3.82019	3.82579	3.79909
5	4.73386	4.76838	4.73113
6	5.6269	5.63331	5.63069
7	6.58144	6.55404	6.58327
8	7.34935	7.51272	7.56096
9	6.3549	6.43567	6.81581
10	7.09609	7.08175	7.2056
11	7.55148	7.51435	7.55124
12	8.18654	8.17937	8.2329
13	8.32659	8.44928	8.52439
14	8.7341	8.87673	9.03532
15	8.89369	9.29693	9.37393
16	9.61221	9.71774	9.75721

Cuadro 4.6: Aceleración obtenida en el Equipo2.

4.8. Análisis de Resultados

El la Figura 4.1 se muestra una gráfica basada en los resultados mostrados en las Tabla 4.2. En esta se puede observar una mejora significativa en el tiempo de codificación con LAME al cambiar de arquitectura, aun así el tiempo de codificación en cada equipo sigue dependiento únicamente del tamaño del archivo..

Tomando la primera fila de las Tablas 4.5 y 4.6 se puede observar que cuando se usa un solo hilo, es decir que el proceso de codificación fue secuencial, se logra una leve mejoria en el tiempo de codificación, esto se debe a que se hace un mejor uso de los recursos porque el proceso de codificación se asigna a un núcleo y la función principal del programa se asigna a otro.

El promedio de aceleración obtenido utilizando i hilos se calcula a través de la Ecuación 4.2, el porcentaje de ganacia en tiempo utilizanzo i hilos se calcula a través de la Ecuación 4.3.

$$A_{Promedio}[i] = \frac{A[i]_{Archivo1} + A[i]_{Archivo2} + A[i]_{Archivo3}}{3} \tag{4.2}$$

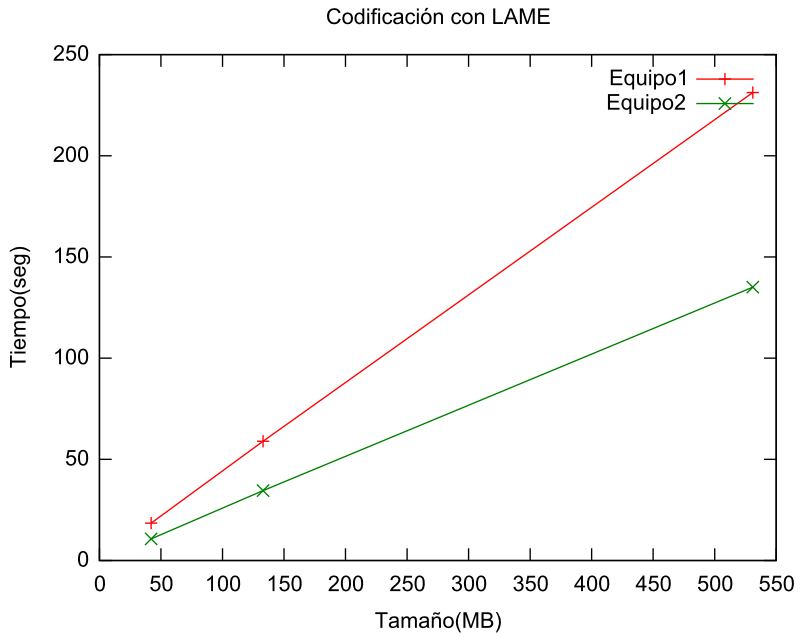

Figura 4.1: Tiempos de Codificación con LAME en Equipo 1 y 2.

$$PG_{Promedio}[i] = (1 - \frac{1}{A_{Promedio}[i]}) \times 100 \tag{4.3}$$

En base a las Tablas 4.3 y 4.4 los resultados obtenidos en el Equipo 1 y 2 son satisfactorios, ya que el primero con 2 hilos nos muestra una aceleración promedio de 1.911, en otras palabras se tiene una ganacia en tiempo del 47.67 % respecto a la codificación secuencial de LAME, el segundo con 16 hilos nos muestra una aceleración prodedio de 9.695 lo cual nos da una ganacia en tiempo del 89.68 % respecto a la codificación secuencial de LAME. Cabe destacar que en la tabla 4.4 el tiempo de codificación del Archivo1 apartir de 7 hilos, el tiempo baja de forma muy ligera esto se debe a que las partes secuenciales del programa como son la unión de los resultados dependen unicamnete del tamaño del archivo de entrada.

En las Figuras 4.2, 4.3 y 4.4 se muestran una serie de gráficas basadas en los resutados de la Tabla 4.4, en la grafícas se observa como el tiempo de codificación disminuye notablemente al aumentar el número hilos, también se observa que en algunos puntos el tiempo se eleva levemente esto se debe a la administración que le da el sistema operativo a los hilos.

Si comparamos las gráficas de las Figuras 4.2, 4.3 y 4.4 podemos observar que tienen un comportamiento casi identico, teniendo como resultado que el tiempo de codificación ya no solo depende del tamaño del archivo sino también del número de hilos

ocupados para la codificación, tomando en cuenta que el número de hilos utilizados debe ser menor o igual al número de núcleos de nuestra arquitectura.

En las pruebas realizadas se utilizo la aplicación ***gnome-system-monitor***, la cual nos muestra el uso de los recursos del sistema, a lo largo de las pruebas se podia observar que el número de hilos era igual al número de núcleos utilizados por el sistema. En contraste la codificación secuencial de LAME mostraba el uso de un solo núcleo.

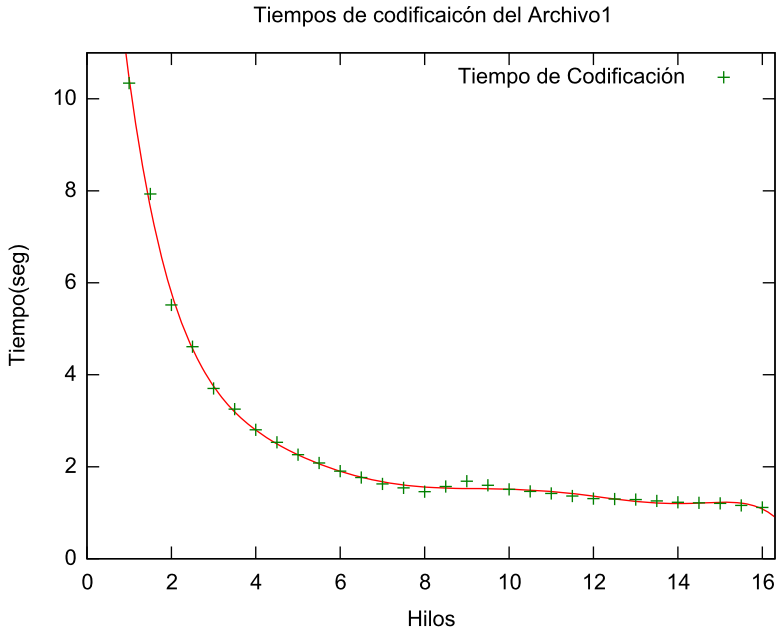

Figura 4.2: Tiempos de Codificación del Archivo1 en Equipo2.

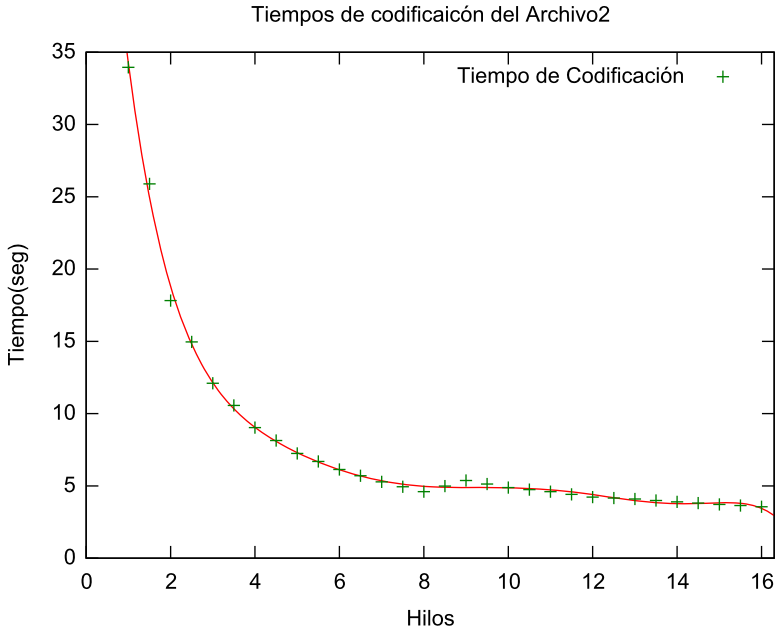

Figura 4.3: Tiempos de Codificación del Archivo2 en Equipo2.

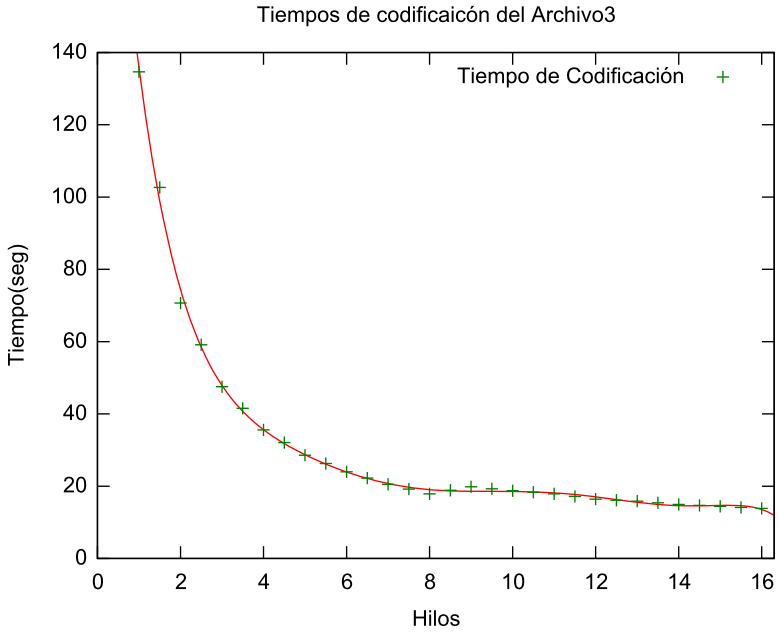

Figura 4.4: Tiempos de Codificación del Archivo3 en Equipo2.

Capítulo 5

Concluciones y Trabajo Futuro

5.1. Concluciones

La programación secuencial tiene muchas ventajas entre ellas, que el desarrollo de software es sencillo y no requiere de un análisis mas allá del de resolver el problema que se esta atacando dejando aun lado las características de las nuevas arquitecturas, obteniendo así una solución con un rendimiento menor en comparación con una solución ya sea concurrente y/o paralela. Al analizar un problema con mayor profundiad nos podría permitir diseñar, desarrollar e implementar soluciones paralelas y/o concurrentes mediante la programación multi-hilo, estas soluciones no solo nos llevan a una ganacia en tiempo sino también en el aprovechamiento de los recursos de las nuevas arquitecturas multi-núcleo. El análisis y diseño de soluciónes paralelas y/o cuncurrentes puede llegar a ser muy complejo debido a que existen problemas donde la solución depende de resultados anteriores, que es donde este tipo de soluciones pueden o no ser adecuadas, esto dependera de las condiciones de competencia que se generen y el manejo de las mismas.

Los resultados presentados en el Capítulo 4, muestran que es posible mejorar el rendimiento de una solución secuencial de la codificación de audio *WAV* a *MP3*, al procesar la codificación de forma concurente mediante la programación multi-hilo. Este tipo de procesamiento nos da como resultado una mejoria notable en el rendimiento en comparación con la solución secuencial. También se comprobo que el uso de los recursos del sistema y el tiempo de codificación es proporcional al número de hilos utilizados.

El particionamiento que se hizo el cual se muestra en la Figura 3.1 se baso en dividir el problema de tal forma que el proceso de codificación se llevara a cabo de forma concurrente sin generar condiciones de competencia. Dicho particionamiento se baso también en que el software secuencial se compone principalmente de tres partes: lectura de datos, procesamiento y escritura de resultado. La lectura y escritura de datos toman un tiempo proporcinal al del tamaño del archivo a codificar, sin embargo

el procesamiento de los datos es el que ocupa la mayor parte del tiempo, dicho tiempo también aumenta de forma proporcinal al tamaño del archivo a codificar. Entonces la parte del procesamiento fue la que se tomo para que se realizara de manera concurrente mediante la programación muti-hilo, teniendo así una solución la cual no solo depende del tamaño de la entrada si no que también de la arquitectura en la cual se esta ejecutando.

Un diseño paralelo y/o concurrente inadecuado podria penalizar en gran parte el redimiento de la solución, si no es que hasta la obtención de resultados inesperados así como un mal uso de los recursos del sistema.

5.2. Trabajo Futuro

El problema de como implementar soluciones paralelas y/o concurrentes de forma eficiente, es aún un problema abierto. Pero simpre se debe tomar en cuenta que este tipo de solucines simpre existe una parte secuencial y otra paralela. Es posible hacer un análisis mas profundo del problema el cual nos llevaria a encontarar la funcionalidad que consume mas recursos y por lo tanto seria propensa a paralelizarse. Lo anterior no solo nos llevaria a obtener ganancias en tiempo superiores si no tambien llegar a aprovechar al máximo las nuevas arquitecturas. Se podria obtener un redimento mayor combinando el procesamiento multi-núcleo y distribuido. Por ultimo se prodria combinar el procesamiento paralelo con hardware dedicado mediante circuitos reprogramables para obtener una mayor ganacia en tiempo y optimización de recursos.

Con base a lo anterior, explorar las posibilidades del diseño paralelo y/o concurrente son muchas. Comentaremos algunas. La primera alternativa es agregar soporte para otros tipos de codificaciones y/o hacer posible la modificación de los parametros de codificación, adaptando la solución presentada. Esto volveria al software desarrollado mas robusto en terminos de formatos soportados. La segunda alternativa es realizar un análisis mas a fondo del codificador *MP3* para encontrar las fucionalidades mas costosas y recurrentes de la solución secuencial, y apartir de esta diseñar soluciones que procesen dichas funcionalidades de forma concurrente, esto dependería de la profundidad del análisis realizado. La tercera alternatva sería llevar la solución a una combinación tanto de procesamiento multi-núcleo y distribuido, obtenieno de esta forma una solución aún mas escalable. Por ultimo otra alternativa es realizar una analisis de las funcionalidades mas recurrentes y costosas de la codificación, y a partir de este generar un co-diseño hardware-software, es decir particionar la solución en software dividiendo las funcionalidades mas costosas, y apartir de esto diseñar una implementación en hardware de dichas funcionalidades, este tipo de particionamiento daría como resultado un sistema de colaboración entre el hardware y software en combinación con el procesamiento multi-núcleo presentado en este trabajo, con lo cual se obtendría un rendimiento mayor. Esto es posible por que el diseño, desarrollo e implementación de sistemas hardware-software es ahora más accesible cuando

se implanta en un circuito programable que cuente con procesadores programados o incrustados.

Apéndice A

Código fuente de los módulos desarrolados

A continuación se muestra en código fuente de la biblioteca desarrollada(libcca.a) la cual integra los módulos de reconocimiento, división, codificación y unión. La codificación se baso en LAME 3.98.2[7] el cual esta licenciado bajo la LGPL[23], por lo cual se puede modificar y ditribuir con la unica condición de hacer referencia a su sitio. Las fuentes de LAME se puden obtener en `http://lame.sourceforge.net`. Se resalta de color rojo el código de LAME(libmp3lame.a).

A.1. cca-threads.h

Definición de las estructuras y funciones para los pthreads.

```
1  #ifndef _CCA_THREADS_H_
2  #define _CCA_THREADS_H_
3
4  #ifndef _GNU_SOURCE
5    #define _GNU_SOURCE
6  #endif
7
8  #include <stdlib.h>
9  #include <string.h>
10 #include <unistd.h>
11 #include <linux/limits.h>
12 #include <pthread.h>
13
14 /*+ ############################################ */
15 /*+LIB LAME (http://lame.sourceforge.net/)*/
16 #include "lame.h"
```

```
17 /*+ ################################################## */
18
19 #include "cca-format.h"
20 #include "cca-recognition.h"
21 #include "cca-parse.h"
22
23
24 /**Estructura para los argumentos para los pthreads de
      codificacion*/
25 typedef struct thread_encoder_struct
26 {
27   int id;
28   char *inPath;
29   char *outPath;
30   int bitwidth;
31   int start;
32   int end;
33   lame_global_flags *gfl;/*+parametros para LAME*/
34 }thread_encoder;
35
36 /**Estructura para los argumentos para los pthreads de inicio
      */
37 typedef struct thread_file_struct
38 {
39   int partition;/**numero de particiones*/
40   char inPath[PATH_MAX+1];
41   char outPath[PATH_MAX+1];
42   cca_input_format *inf;
43 }thread_file;
44
45 /** thread para cada archivo de entrada*/
46 void *thread_function_file(void *args);
47
48 /*Manejo global de CCA*/
49
50 /**Estructura global del programa de linea de comandos*/
51 typedef struct cca_global_struct
52 {
53   int cores;  /*numero de cores de la arquitectura*/
54   int in_files; /*numero de archivos de entrada*/
55   /*Variables para el manejo de los pthread*/
56   pthread_t *thread_id;
57   thread_file *thread_arg;
```

```
58 }cca_global_flags;
59
60 cca_global_flags * cca_init_gf();
61 int cca_init_params_gf(cca_global_flags *gfp);
62 void cca_close_gf(cca_global_flags * gfp);
63 char * cca_init_outfile(char * inPath,char  * outPath, int id
     );
64 int cca_get_num_cores();
65 int cca_init_out_format(cca_input_format *inf,
     lame_global_flags *gfl);
66
67 #endif
```

A.2. cca-threads.c

Funciones para el manejo de los pthreads.

```
1
2 #include "cca-threads.h"
3 #include "cca-partition.h"
4 #include "cca-get_audio.h"
5
6 void *thread_function_encoder(void *args);
7 void cca_union_mp3(thread_file *tf,thread_encoder *te);
8
9 /*#########################################################*/
10 /*Manejo global de CCA*/
11 /*#########################################################*/
12
13 /**Regresa el numero de nucleos de la arquitectura*/
14 int cca_get_num_cores()
15 {
16   char *cad;
17   FILE *cpufile;
18   size_t nbytes=2048;
19   int cores=0;
20   cpufile=fopen("/proc/cpuinfo","r");
21   if(cpufile!=NULL)
22   {
23     cad=(char *)malloc(sizeof(char)*nbytes);
24     while(getline(&cad,&nbytes,cpufile)!=-1)
25     {
```

```
26          if(strncmp("processor",cad,9)==0)
27            ++cores;
28          free(cad);
29          cad=(char *)malloc(sizeof(char)*nbytes);
30        }
31        fclose(cpufile);
32      }
33      else
34      {
35        printf("Error al abrir /proc/cpuinfo\n");
36        return -1;
37      }
38      if(cores<=0)
39      {
40        printf("/proc/cpuinfo corrupto\n");
41        return -1;
42      }
43      return cores;
44  }
45
46  /**Inicializacion de cca_global_flags del programa*/
47  cca_global_flags * cca_init_gf()
48  {
49        cca_global_flags * gfp =(cca_global_flags *)calloc(1,
              sizeof(cca_global_flags));
50        if (gfp == NULL)
51            gfp=NULL;
52      gfp->cores=cca_get_num_cores();
53      gfp->in_files=1;
54      gfp->thread_id=NULL;
55      gfp->thread_arg=NULL;
56        if (gfp->cores<= 0 || gfp==NULL)
57        {
58          printf("Error durante la inicializacion\n");
59            free(gfp);
60            gfp=NULL;
61        }
62        return gfp;
63  }
64
65  /**Inicializacion de los parametros de cca_global_flags*/
66  int cca_init_params_gf(cca_global_flags *gfp)
67  {
```

```
68    /*Depende del numero de archivos de entrada*/
69    gfp->thread_id=(pthread_t *)calloc(gfp->in_files,sizeof(
          pthread_t));
70    gfp->thread_arg=(thread_file *)calloc(gfp->in_files,sizeof(
          thread_file));
71
72    if(gfp->thread_id==NULL || gfp->thread_arg==NULL)
73    {
74      printf("Error durante la inicializacion de parametros\n")
            ;
75      free(gfp);
76      return -1;
77    }
78    return 0;
79  }
80
81  /** Libera los recursos de cca_global_flags*/
82  void cca_close_gf(cca_global_flags * gfp)
83  {
84    if(gfp)
85    {
86      if(gfp->thread_id!=NULL) free(gfp->thread_id);
87      if(gfp->thread_arg!=NULL) free(gfp->thread_arg);
88      gfp->thread_id=NULL;
89      gfp->thread_arg=NULL;
90      free(gfp);
91      //gfp=NULL;
92    }
93  }
94
95  /** Creacion de los nombres temporales*/
96  char * cca_init_outfile(char * inPath,char  * outPath, int id
        )
97  {
98    int len;
99    len=strlen(inPath);
100   outPath=(char *)calloc(len+4,sizeof(char));
101   strncpy(outPath,inPath,len);
102   outPath[len-3]='m';
103   outPath[len-2]='p';
104   outPath[len-1]='3';
105   outPath[len]=id/100+48;
106   outPath[len+1]=((id%100)/10)+48;
```

```
107   outPath[len+2]=((id%100)%10)+48;
108   outPath[len+3]='\0';
109   return outPath;
110 }
111
112 /**Inicializacion de los parametros para la codificacion con
      LAME*/
113 int cca_init_out_format(cca_input_format *inf,
      lame_global_flags *gfl)
114 {
115   int ret=0;
116   switch(inf->sff)
117   {
118     case cca_sf_wave:
119       /*+#########################################*/
120       /*+Envio de los parametros para codificar con LAME*/
121       lame_set_num_channels(gfl, inf->sf.riff.Channels);
122       lame_set_in_samplerate(gfl, inf->sf.riff.Samplerate);
123       lame_set_num_samples(gfl, inf->sf.riff.Data_Size / (inf
            ->sf.riff.Channels * ((inf->sf.riff.Bits_Per_Sample
            + 7) / 8)));
124       ret=lame_init_params(gfl);
125       /*+#########################################*/
126       break;
127     default:
128       break;
129   }
130   return ret;
131 }
132
133 /*################################################*/
134 /*Codificacion paralela*/
135 /*################################################*/
136
137 /** thread para cada archivo de entrada*/
138 void *thread_function_file(void *args)
139 {
140   thread_file *arg=(thread_file *)args;
141   int i;
142   pthread_t *thread_id;
143   thread_encoder *thread_arg;
144
```

```
145   thread_id=(pthread_t *)calloc(arg->partition,sizeof(
          pthread_t));
146   thread_arg=(thread_encoder *) calloc(arg->partition,sizeof(
          thread_encoder));
147   for(i=0;i<arg->partition;i++)
148   {
149     thread_arg[i].id=i;
150     thread_arg[i].inPath=arg->inPath;
151     if(arg->partition>1)
152     {
153       thread_arg[i].outPath=cca_init_outfile(arg->inPath,
              thread_arg[i].outPath,i);
154     }
155     else
156     {
157       thread_arg[i].outPath=(char *)calloc(strlen(arg->
              outPath)+1,sizeof(char));
158       strncpy(thread_arg[i].outPath,arg->outPath,strlen(arg->
              outPath));
159     }
160     thread_arg[i].gfl=lame_init();
161     cca_init_out_format(arg->inf,thread_arg[i].gfl);
162     thread_arg[i].bitwidth=arg->inf->sf.riff.Bits_Per_Sample;
163     if (arg->inf->sff==cca_sf_wave)
164       cca_partition_wav(i,&(arg->inf->sf.riff),arg->partition
              ,&(thread_arg[i].start),&(thread_arg[i].end));
165     pthread_create(&thread_id[i], NULL,
            thread_function_encoder,(void *)&thread_arg[i]);
166   }
167
168     for(i=0;i<arg->partition;i++)
169     {
170       pthread_join(thread_id[i],NULL);
171     lame_close(thread_arg[i].gfl);
172     }
173
174   /*union*/
175   if(arg->partition>1)
176     cca_union_mp3(arg,thread_arg);
177   /*termina union*/
178   free(thread_id);
179   free(thread_arg);
180   pthread_exit(NULL);
```

```
181 }
182
183 /** Hilo de codifiacion*/
184 void *thread_function_encoder(void *args)
185 {
186   int iread=0,remaining,imp3=0,owrite,error=0;
187   int Buffer[2][1152];
188   unsigned char mp3buffer[LAME_MAXMP3BUFFER];
189   thread_encoder *arg=(thread_encoder *)args;
190   FILE *outfile=NULL,*infile=NULL;
191   lame_global_flags *gfl=arg->gfl;
192   int channels=lame_get_num_channels(gfl);
193   infile=fopen(arg->inPath,"rb");
194   remaining=arg->end-arg->start;
195
196   if(infile!=NULL)
197   {
198     outfile=fopen(arg->outPath,"w+b");
199     fseek(infile,arg->start,SEEK_SET);
200     do
201     {
202       iread = cca_get_audio(gfl, infile,remaining,Buffer,arg
          ->bitwidth);
203
204       if(iread>=0)
205       {
206         /* codificacion*/
207         /*+##############################################*/
208         /*+Llamada a la funcion de codificacion de LAME*/
209         imp3 = lame_encode_buffer_int(arg->gfl,Buffer[0],
            Buffer[1],iread,mp3buffer,sizeof(mp3buffer));
210         /*+##############################################*/
211         remaining-=iread*channels*(arg->bitwidth/8);
212
213         /* was our output buffer big enough? */
214         if (imp3 < 0)
215         {
216           if (imp3 == -1)
217             fprintf(stderr,"mp3 buffer is not big enough ... \
                n");
218           else
219             fprintf(stderr,"mp3 internal error:  error code=%
                i\n", imp3);
```

```
220             error=-1;
221           }
222           else
223           {
224             owrite = (int) fwrite(mp3buffer, 1, imp3, outfile);
225             if (owrite != imp3) {
226               fprintf(stderr,"Error writing mp3 output \n");
227               error=-1;
228             }
229           }
230         }
231       } while (remaining>0 && iread>0 && error>=0);
232
233       fclose(infile);
234       fclose(outfile);
235       }
236     pthread_exit(NULL);
237 }
238
239 /*###############################################*/
240 /*   Union                                       */
241 /*###############################################*/
242
243 /**Union de las partes codificadas*/
244 void cca_union_mp3(thread_file *tf,thread_encoder *te)
245 {
246   int i;
247   unsigned char buffer[LAME_MAXMP3BUFFER];
248   size_t iread,owrite;
249   FILE *outfile=NULL,*infile=NULL;
250
251   outfile=fopen(tf->outPath,"w+b");
252   for(i=0;i<tf->partition;i++)
253   {
254     infile=fopen(te[i].outPath,"rb");
255     iread=fread(buffer, sizeof(unsigned char),
          LAME_MAXMP3BUFFER, infile);
256     while(iread>0)
257     {
258       owrite=fwrite(buffer,sizeof(unsigned char),iread,
            outfile);
259       iread=fread(buffer, sizeof(unsigned char),
            LAME_MAXMP3BUFFER, infile);
```

```
260       }
261       fclose(infile);
262       infile=NULL;
263       unlink(te[i].outPath);
264       free(te[i].outPath);
265       }
266     fclose(outfile);
267 }
```

A.3. cca-format.h

Estructuras de los formatos de entrada y salida soportados.

```
1
2 #ifndef _CCA_FORMAT_H_
3 #define _CCA_FORMAT_H_
4
5 #include<stdio.h>
6 #include<stdint.h>
7
8 /** Modos MPEG */
9 typedef enum cca_MPEG_mode_e {
10   cca_JOINT_STEREO,
11   cca_MONO,
12   cca_NOT_SET
13 } cca_MPEG_mode;
14
15 /*----------------------Formatos de entrada----------------------
     */
16 static int const WAV_ID_RIFF = 0x52494646; /* "RIFF" */
17 static int const WAV_ID_WAVE = 0x57415645; /* "WAVE" */
18 static int const WAV_ID_FMT = 0x666d7420; /* "fmt " */
19 static int const WAV_ID_DATA = 0x64617461; /* "data" */
20 static short const WAVE_FORMAT_PCM = 0x0001;
21
22 /**Estructura del formato RIFF*/
23 typedef struct cca_riff_struct
24 {
25   int Riff_Id;
26   int Riff_Size;
27   int Format;
28   int Fmt_Id;
```

```
29    int Fmt_Size;
30    short Audio_Format;
31      short Channels;
32      int Samplerate;
33      int Byterate;
34      short Block_Align;
35      short Bits_Per_Sample;
36      int Data_Id;
37      int Data_Size;
38      int NumSamples;
39  }cca_riff_format;
40
41  /**Archivos de audio soportados*/
42  typedef enum cca_sound_format_struct {
43      cca_sf_unknown,
44      cca_sf_wave,
45  } cca_sound_file_format;
46
47  /**Union que contiene las estructuras de los formatos
       soportados*/
48  typedef union cca_format_union
49  {
50    cca_riff_format riff;
51  }cca_union_format;
52
53  /**Estructura global del formato de entrada*/
54  typedef struct cca_input_struct
55  {
56    cca_sound_file_format sff;
57    cca_union_format sf;
58  }cca_input_format;
59
60  void cca_print_format(cca_input_format * const inf);
61
62  #endif
```

A.4. cca-format.c

Funciones para los formatos de entrada y salida soportados.

```
1   #include "cca-format.h"
2
```

```
3
4  void cca_print_wave_format(cca_input_format * const inf);
5
6  /**
7    Muestra las caracteristicas del formato de entrada
8  */
9  void cca_print_format(cca_input_format * const inf)
10 {
11   switch(inf->sff)
12   {
13     case cca_sf_wave:
14       cca_print_wave_format(inf);
15       break;
16     default:
17       break;
18   }
19 }
20
21 /**
22   Muestra las atributos del formato Wave
23 */
24 void cca_print_wave_format(cca_input_format * const inf)
25 {
26   printf("Archivo tipo %d\n",inf->sff);
27   printf("id_riff: %d\n",inf->sf.riff.Riff_Id);
28   printf("riif_size: %d\n",inf->sf.riff.Riff_Size);
29   printf("riff_format: %d\n",inf->sf.riff.Format);
30   printf("id_fmt: %d\n",inf->sf.riff.Fmt_Id);
31   printf("fmt_size: %d\n",inf->sf.riff.Fmt_Size);
32   printf("audio_format: %d\n",inf->sf.riff.Audio_Format);
33   printf("channels: %d\n",inf->sf.riff.Channels);
34   printf("samplerate: %d\n",inf->sf.riff.Samplerate);
35   printf("byterate: %d\n",inf->sf.riff.Byterate);
36   printf("block_aling: %d\n",inf->sf.riff.Block_Align);
37   printf("bits_per_sample: %d\n",inf->sf.riff.Bits_Per_Sample
       );
38   printf("id_data: %d\n",inf->sf.riff.Data_Id);
39   printf("data_size: %d\n",inf->sf.riff.Data_Size);
40   printf("numsamples: %d\n",inf->sf.riff.NumSamples);
41 }
```

A.5. cca-get_audio.h

Definición de las funciones para la lectura de muestras de audio.

```
1  #ifndef _CCA_GET_AUDIO_H_
2  #define _CCA_GET_AUDIO_H_
3
4  int cca_Read16BitsLowHigh(FILE * fp);
5  int cca_Read16BitsHighLow(FILE * fp);
6  int cca_Read32BitsLowHigh(FILE * fp);
7  int cca_Read32BitsHighLow(FILE * fp);
8  int cca_get_audio(lame_global_flags * const gfl,FILE *fd,int
       remaining ,int buffer[2][1152],int bitwidth);
9
10 #endif
```

A.6. cca-get_audio.c

Funciones para la lectura de muestras de audio.

```
1  #include <inttypes.h>
2  #include <stdio.h>
3  #include <string.h>
4
5
6  #include "cca-threads.h"
7  #include "cca-get_audio.h"
8
9  static int cca_read_samples_pcm(FILE * musicin, int
       sample_buffer[2304], int samples_to_read,int pcmbitwidth);
10 static int cca_unpack_read_samples(const int samples_to_read,
        const int bytes_per_sample, const int swap_order, int *
       sample_buffer, FILE * pcm_in);
11
12
13 /*------------------------------
14  * Big/little-endian I/O
15 ------------------------------*/
16 /**Lectura de 16bits LowHigh*/
17 int cca_Read16BitsLowHigh(FILE * fp)
18 {
19     short result=0;
```

```
20    fread(&result,2,1,fp);
21      return ((int)result);
22  }
23
24  /**Lectura de 16bits HighLow*/
25  int cca_Read16BitsHighLow(FILE * fp)
26  {
27      int      first, second, result;
28      first = 0xff & getc(fp);
29      second = 0xff & getc(fp);
30      result = (first << 8) + second;
31
32      return (result);
33  }
34
35  /**Lectura de 32bits LowHigh*/
36  int cca_Read32BitsLowHigh(FILE * fp)
37  {
38      int result=0;
39    fread(&result,4,1,fp);
40      return (result);
41  }
42
43  /**Lectura de 32bits HighLow*/
44  int cca_Read32BitsHighLow(FILE * fp)
45  {
46      int      first, second, result;
47
48      first = 0xffff & cca_Read16BitsHighLow(fp);
49      second = 0xffff & cca_Read16BitsHighLow(fp);
50      result = (first << 16) + second;
51      return (result);
52  }
53
54
55  /** Lee un muestras de audio del archivo en el buffer,  a
       linea los datos para un posterior procesamiento, y separa
       los canales*/
56  int cca_get_audio(lame_global_flags * const gfl,FILE *fd, int
       remaining ,int buffer[2][1152],int bitwidth)
57  {
58    int insamp[2 * 1152];
59    int samples_read=0,samples_to_read=0,bytes_to_read;
```

```
60    int i;
61    int *p;
62    int channels = lame_get_num_channels(gfl);
63    int framesize = lame_get_framesize(gfl);
64
65    bytes_to_read=channels * framesize* (bitwidth/8);
66    samples_to_read=remaining>bytes_to_read?bytes_to_read:
         remaining;
67
68    samples_read = cca_read_samples_pcm(fd, insamp, channels *
         framesize,bitwidth);
69
70      if (samples_read < 0)
71          return samples_read;
72
73      p = insamp + samples_read;
74      samples_read /= channels;
75      if (channels == 2) {
76          for (i = samples_read; --i >= 0;) {
77              buffer[1][i] = *--p;
78              buffer[0][i] = *--p;
79          }
80      }
81      else if (channels == 1) {
82          memset(buffer[1], 0, samples_read * sizeof(int));
83          for (i = samples_read; --i >= 0;) {
84              buffer[0][i] = *--p;
85          }
86      }
87      return samples_read;
88  }
89
90  /**Lee muestras de audio PCM del archivo de entrada al buffer
       */
91
92  static int cca_read_samples_pcm(FILE * musicin, int
       sample_buffer[2304], int samples_to_read,int pcmbitwidth)
93  {
94      int      samples_read=0;
95      switch (pcmbitwidth)
96    {
97      case 32:
98      case 24:
```

```
99      case 16:
100       samples_read = cca_unpack_read_samples(samples_to_read,
              pcmbitwidth / 8,0,sample_buffer,musicin);
101         break;
102     case 8:
103       samples_read = cca_unpack_read_samples(samples_to_read
              ,1, 1, sample_buffer,musicin);
104         break;
105     }
106     if (ferror(musicin)) {
107         printf("Error al leer el archivo de entrada %d\n",
                samples_read);
108         samples_read=0;
109     }
110
111     return samples_read;
112 }
113
114 /**Lee muestras de audio PCM archivo de entrada a el buffer
      tomando en cuenta el orden de los bytes*/
115 static int cca_unpack_read_samples(const int samples_to_read,
       const int bytes_per_sample, const int swap_order, int *
      sample_buffer, FILE * pcm_in)
116 {
117     size_t  samples_read;
118     int     i;
119     int    *op;               /* puntero de salida */
120     unsigned char *ip = (unsigned char *) sample_buffer;
121     const int b = sizeof(int) * 8;
122
123
124 #define GA_URS_IFLOOP( ga_urs_bps ) \
125     if( bytes_per_sample == ga_urs_bps ) \
126
127   for( i=samples_read*bytes_per_sample; (i -=
        bytes_per_sample)>=0;)
128
129   samples_read = fread(sample_buffer, bytes_per_sample,
        samples_to_read, pcm_in);
130
131   op = sample_buffer + samples_read;
132
133     if (swap_order == 0){
```

```
134             GA_URS_IFLOOP(1)
135             *--op = ip[i] << (b - 8);
136             GA_URS_IFLOOP(2)
137             *--op = ip[i] << (b - 16) | ip[i + 1] << (b - 8);
138             GA_URS_IFLOOP(3)
139             *--op = ip[i] << (b - 24) | ip[i + 1] << (b - 16) |
                    ip[i + 2] << (b - 8);
140             GA_URS_IFLOOP(4)
141             *--op = ip[i] << (b - 32) | ip[i + 1] << (b - 24) |
                    ip[i + 2] << (b - 16) | ip[i + 3] << (b - 8);
142         }
143         else {
144             GA_URS_IFLOOP(1)
145             *--op = (ip[i] ^ 0x80) << (b - 8) | 0x7f << (b - 16)
                    ;
146             GA_URS_IFLOOP(2)
147             *--op = ip[i] << (b - 8) | ip[i + 1] << (b - 16);
148             GA_URS_IFLOOP(3)
149             *--op = ip[i] << (b - 8) | ip[i + 1] << (b - 16) |
                    ip[i + 2] << (b - 24);
150             GA_URS_IFLOOP(4)
151             *--op = ip[i] << (b - 8) | ip[i + 1] << (b - 16) |
                    ip[i + 2] << (b - 24) | ip[i + 3] << (b - 32);
152         }
153
154 #undef GA_URS_IFLOOP
155         return (samples_read);
156 }
```

A.7. cca-parse.h

Definición de las funciones para el análisis de los parámetros del programa.

```
1 #ifndef _CCA_PARSE_H_
2 #define _CCA_PARSE_H_
3
4 #include<stdio.h>
5 #include<string.h>
6
7 void cca_usage(const char *ProgramName);
8 void cca_help(const char *ProgramName);
9 int  cca_parse_args(int argc, char *argv[], char * const
     inPath, char * const outPath,int *n);
```

```
10 void cca_print_version();
11
12 #endif
```

A.8. cca-parse.c

Funciones para el análisis de los parámetros del programa.

```
1
2 #include "cca-threads.h"
3 #include "cca-version.h"
4 #include "cca-parse.h"
5
6
7 /**Muestra la version del programa*/
8 void cca_print_version()
9 {
10     const char *b = cca_get_os_bitness();
11     const char *v = cca_get_version();
12     const char *u = cca_get_url();
13     const size_t lenb = strlen(b);
14     const size_t lenv = strlen(v);
15     const size_t lenu = strlen(u);
16     const size_t lw = 80;        /* caracteres maximos en
           consola */
17     const size_t sw = 16;        /* caracteres minimos en
           consola */
18
19   if (lw >= lenb + lenv + lenu + sw || lw < lenu + 2)
20   {
21     if (lenb > 0)
22       printf("CCA %s version %s (%s)\n\n", b, v, u);
23     else
24       printf("CCA version %s (%s)\n\n", v, u);
25   }
26   else
27   {
28     if (lenb > 0)
29       printf("CCA %s version %s\n%*s(%s)\n\n", b, v, lw - 2 -
             lenu, "", u);
30     else
31       printf("CCA version %s\n%*s(%s)\n\n", v, lw - 2 - lenu,
             "", u);
```

```
32    }
33  }
34
35  /** Muestra el la sintaxis de uso del programa*/
36  void cca_usage(const char *ProgramName)
37  {
38      printf("uso: %s [options] <infile> [outfile]\n\n"
39             "Pruebe:\n"
40          "\t\" %s -?\"\t\tPara ayda\n\n",
41          ProgramName, ProgramName);
42  }
43
44
45  /** Muestra la ayuda del programa*/
46  void cca_help(const char *ProgramName)
47  {
48    int cores=cca_get_num_cores();
49    printf("uso: %s [opciones] <infile> [outfile]\n",
        ProgramName);
50    printf("OPCIONES:\n"
51               "    -n num       Numero de particiones (n<= %d && n
                     > 0)\n"
52               "                 Recomendado n= %d\n"
53               "    -?           Lista de opciones\n" "\n"
54               ,cores,cores);
55  }
56
57  /** Analisis de los argumentos de programa*/
58  int cca_parse_args(int argc, char *argv[],char * const inPath
       , char * const outPath,int *n)
59  {
60      int input_file = 0;
61      int i;
62      size_t size=0;
63      const char *ProgramName = argv[0];
64
65      inPath[0] = '\0';
66      outPath[0] = '\0';
67
68      /* procesando argumentos */
69     for (i = 0; ++i < argc;)
70    {
71          char     c;
```

```
72          char    *token;
73          char    *arg;
74          char    *nextArg;
75          int      argUsed;
76
77          token = argv[i];
78      if (*token++ == '-')
79      {
80        argUsed = 0;
81        nextArg = (char *)(i + 1 < argc ? argv[i + 1] : "");
82        while ((c = *token++) != '\0')
83        {
84          arg = *token ? token : nextArg;
85          switch (c)
86          {
87            case '?':
88              cca_help(ProgramName);
89              return -2;
90              break;
91            case 'n':
92              *n=atoi(argv[++i]);
93              if((*n)<=0)
94              {
95                fprintf(stdout,"%s: -n %d, n debe ser mayor que
                      0\n",ProgramName,*n);
96                return -1;
97              }
98              break;
99            default:
100             fprintf(stdout,"%s: opcion no reconocida - %c\n",
                    ProgramName, c);
101             return -1;
102         }
103         if (argUsed)
104         {
105           if (arg == token)
106             token = (char *)""; /* no more from token */
107           else
108             ++i; /* skip arg we used */
109           arg = (char *)"";
110           argUsed = 0;
111         }
112       }
```

```
113       }
114       else
115       {
116         /* opciones normales:    inputfile   [outputfile]*/
117         if (inPath[0] == '\0')
118         {
119           size=strlen(argv[i]);
120           strncpy(inPath, argv[i], size);
121           inPath[size+1]='\0';
122           input_file = 1;
123         }
124         else
125         {
126           if (outPath[0] == '\0')
127           {
128             size=strlen(argv[i]);
129             strncpy(outPath, argv[i], size);
130             outPath[size+1]='\0';
131           }
132           else
133           {
134             printf("%s: argumento extra invalido \"%s\"\n",
                    ProgramName, argv[i]);
135             return -1;
136           }
137         }
138       }
139     } /*termina la verificacion de opciones */
140
141     if (!input_file)
142     {
143       cca_usage(ProgramName);
144       return -1;
145     }
146
147     if (outPath[0] == '\0')
148     {
149       size=strlen(inPath);
150       strncpy(outPath, inPath, size-4);
151       strncat(outPath, ".mp3", 4);
152       outPath[size+1]='\0';
153     }
154     return 0;
```

```
155 }
```

A.9. cca-partition.h

Biblioteca para la división de archivos de audio.

```
1 #ifndef _CCA_PARTITION_H_
2 #define _CCA_PARTITION_H_
3 #include<stdio.h>
4 #include"cca-format.h"
5
6 void cca_partition_wav(int const id, cca_riff_format const *
    riff_f, int const partition, int *start, int *end);
7 #endif
```

A.10. cca-partition.c

Función para la división de archivos de audio.

```
1 #include"cca-partition.h"
2
3 /** Divide un archivo wav partition veces, creando un inicio
    y un fin para la lectura del archivo*/
4 void cca_partition_wav(int const id,cca_riff_format const *
    riff_f, int const partition,int *start,int *end)
5 {
6   /* El buffer minimo para un frame mp3 de doble canal es de
       1152*2 por los bytes que forman un sample */
7   int framesize=1152*riff_f->Channels*(riff_f->
      Bits_Per_Sample/8);
8   int offset=riff_f->Data_Size/(framesize*partition);
9
10  *start=( id*offset*framesize)+44;/*inicio para la lectura*/
11  if(id!=(partition-1))
12    *end=( (id+1)*offset*framesize)+44; /*fin de lectura*/
13  else
14    *end=riff_f->Data_Size+44; /*fin de lectura*/
15 }
```

A.11. cca-recognition.h

Biblioteca para el reconocimiento de archivos de audio.

```
1 #ifndef _CCA_RECOGNITION_H_
2 #define _CCA_RECOGNITION_H_
3
4 #include"cca-format.h"
5
6 cca_input_format * cca_recognition(char * const inPath);
7
8 #endif
```

A.12. cca-recognition.c

Funciones para el reconocimiento de archivos de audio.

```
1 #include <stdio.h>
2 #include <inttypes.h>
3 #include <math.h>
4
5 #include "cca-threads.h"
6 #include "cca-get_audio.h"
7 #include "cca-recognition.h"
8
9 cca_input_format * cca_parse_file_header(FILE * sf);
10 cca_sound_file_format cca_parse_wave_header(cca_union_format
     *sw, FILE * musicin);
11
12 /**Analisis del encabezado de los archivos de entrada */
13 cca_input_format * cca_recognition(char * const inPath)
14 {
15   cca_input_format * inf=NULL;
16   FILE *musicin;
17   if((musicin = fopen(inPath, "rb")) == NULL)
18     printf("No se encontro \"%s\".\n", inPath);
19   else
20   inf=cca_parse_file_header(musicin);
21   fclose(musicin);
22   return inf;
23 }
24
```

```
25 /**Analisis de encabezados de los archivos de audio
       soportados*/
26 cca_input_format * cca_parse_file_header(FILE * musicin)
27 {
28   cca_input_format * inf=NULL;
29   int type;
30   inf=(cca_input_format *)calloc(1,sizeof(cca_input_format));
31   inf->sff=cca_sf_unknown;
32   type = cca_Read32BitsHighLow(musicin);
33   if (type == WAV_ID_RIFF)
34   {
35     inf->sf.riff.Riff_Id=type;
36     inf->sff=cca_parse_wave_header(&inf->sf, musicin);
37     if (inf->sff==cca_sf_wave)
38       return inf;
39     else
40       printf("Error: archivo wav corrupto\n");
41   }
42   else
43     printf("Error: formato de audio no soportado\n");
44     return NULL;
45 }
46
47 /**
48   Analisis del encabezado Microsoft Wave
49 */
50
51 cca_sound_file_format cca_parse_wave_header(cca_union_format
       *sw, FILE * musicin)
52 {
53   sw->riff.Riff_Size= cca_Read32BitsLowHigh(musicin);
54   sw->riff.Format=cca_Read32BitsHighLow(musicin);
55
56   if (sw->riff.Format == WAV_ID_WAVE)
57   {
58     sw->riff.Fmt_Id = cca_Read32BitsHighLow(musicin);
59     if (sw->riff.Fmt_Id == WAV_ID_FMT)
60     {
61       sw->riff.Fmt_Size = cca_Read32BitsLowHigh(musicin);
62       if (sw->riff.Fmt_Size == 16)
63       {
64         sw->riff.Audio_Format = cca_Read16BitsLowHigh(musicin
             );
```

```
65          sw->riff.Channels = cca_Read16BitsLowHigh(musicin);
66          sw->riff.Samplerate = cca_Read32BitsLowHigh(musicin);
67          sw->riff.Byterate = cca_Read32BitsLowHigh(musicin);
68          sw->riff.Block_Align = cca_Read16BitsLowHigh(musicin)
                ;
69          sw->riff.Bits_Per_Sample = cca_Read16BitsLowHigh(
                musicin);
70
71          if (sw->riff.Audio_Format== WAVE_FORMAT_PCM)
72          {
73              sw->riff.Data_Id = cca_Read32BitsHighLow(musicin)
                    ;
74              if (sw->riff.Data_Id == WAV_ID_DATA)
75              {
76                sw->riff.Data_Size= cca_Read32BitsLowHigh(
                      musicin);
77                if (sw->riff.Channels>0 && sw->riff.Channels<3
                      )
78                {
79                  sw->riff.NumSamples=sw->riff.Data_Size / (sw
                        ->riff.Channels * (sw->riff.
                        Bits_Per_Sample/ 8));
80                  return cca_sf_wave;
81                }
82                else
83                  printf("WAV: Numero de canales invalido\n");
84              }
85              else
86                printf("WAV: Dat_Id incorrecto\n");
87          }
88          else
89            printf("WAV: Formato incorrecto \n");
90
91        }
92        else
93          printf("WAV: fmt_id incorrecto\n");
94      }
95    }
96    return cca_sf_unknown;
97  }
```

A.13. cca-version.h

Biblioteca con los datos de la versión de CCA.

```
1
2 #ifndef _CCA_VERSION_H_
3
4 /** sitio del programa*/
5 #define URL "http://sites.google.com/site/
    cbicomputerengineering/cca"
6 /** version */
7 #define VERSION_MAYOR 1 /* Mayor */
8 /** sub version*/
9 #define VERSION_MENOR 1 /* Menor */
10
11 const char * cca_get_version();
12 const char * cca_get_url();
13 const char * cca_get_os_bitness();
14
15 #endif
```

A.14. cca-version.c

Biblioteca con los datos de la versión de CCA.

```
1 #include<stdio.h>
2 #include<string.h>
3
4 #include"cca-version.h"
5
6 #define STR(x)   #x
7 #define XSTR(x)  STR(x)
8
9 /**Regresa la version del programa*/
10 const char * cca_get_version()
11 {
12     static const char *const str =
13         XSTR(VERSION_MAYOR) "." XSTR(VERSION_MENOR);
14     return str;
15 }
16
17 /**Regresa la url de contanco del programa*/
```

```
18 const char * cca_get_url()
19 {
20     static const char *const str = URL;
21     return str;
22 }
23
24 /**Regresa el tipo de arquitectura 32 o 64 bits*/
25 const char * cca_get_os_bitness()
26 {
27     static const char *const strXX = "";
28     static const char *const str32 = "32bits";
29     static const char *const str64 = "64bits";
30
31     switch (sizeof(void *))
32     {
33     case 4:
34         return str32;
35     case 8:
36         return str64;
37     default:
38         return strXX;
39     }
40 }
```

Apéndice B

Código de la GUI

A continuación se muestra en código fuente de la GUI de la aplicación, la cual fue desarrollada con Qt4[5].

B.1. mainwindow.h

Definición de la clase *MainWindow*, la cual representa la ventana principal.

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QTableWidget>
6  #include <list>
7
8  #include "cca_interface.h"
9
10 namespace Ui {
11    class MainWindow;
12 }
13
14 class MainWindow : public QMainWindow
15 {
16    Q_OBJECT
17
18 public:
19    explicit MainWindow(QWidget *parent = 0);
20    ~MainWindow();
21 private:
```

```
22    Ui::MainWindow *ui;
23    QString host;
24    QString os;
25    QString kernel;
26    cca_interface *icca;
27    void addRow(QTableWidget *table);
28    bool fileExists(QString name);
29  private slots:
30    void on_cmdRemove_clicked();
31    void on_cmdAdd_clicked();
32    void on_cmdStop_clicked();
33    void on_cmdStart_clicked();
34    void on_cmdBuscar_clicked();
35    void on_cmdClean_clicked();
36    void on_actionStop_triggered();
37    void on_actionStart_triggered();
38    void on_actionBrowse_triggered();
39    void on_actionClean_triggered();
40    void on_actionRemove_triggered();
41    void on_actionAdd_triggered();
42    void on_actionExit_triggered();
43    void on_actionAbout_triggered();
44  public:
45    void finishEncode();
46    void checkRow(int i);
47  };
48  #endif // MAINWINDOW_H
```

B.2. mainwindow.cpp

Funciones de la clase *MainWindow*, para el manejo de los eventos de la interfaz.

```
1  #include <QCheckBox>
2  #include <QSpinBox>
3  #include <QTableWidget>
4  #include <QFileDialog>
5  #include <QFileInfo>
6  #include <QMessageBox>
7  #include <sys/utsname.h>
8
9  #include "mainwindow.h"
10 #include "ui_mainwindow.h"
```

```
11 #include "about.h"
12
13 MainWindow::MainWindow(QWidget *parent) :
14     QMainWindow(parent),
15     ui(new Ui::MainWindow)
16
17 {
18     struct utsname machine;
19     icca=new cca_interface(this);
20     ui->setupUi(this);
21     ui->tbl_audio->setColumnWidth(0,395);
22     ui->tbl_audio->setColumnWidth(1,50);
23     ui->tbl_audio->setColumnWidth(2,100);
24     uname(&machine);
25     host=tr(machine.nodename);
26     os=tr("GNU/")+tr(machine.sysname)+tr("␣")+tr(machine.
           machine);
27     kernel=tr("Kenel␣")+tr(machine.release);
28     ui->txtCores->setText(QString::number(icca->getCores()));
29     ui->txtHost->setText(host);
30     ui->txtKernel->setText(kernel);
31     ui->txtOs->setText(os);
32 }
33
34 MainWindow::~MainWindow()
35 {
36     delete this->ui;
37     delete this->icca;
38 }
39
40 void MainWindow::on_cmdAdd_clicked()
41 {
42     if(ui->tbl_audio->rowCount()<16)
43         addRow(ui->tbl_audio);
44 }
45
46 void MainWindow::addRow(QTableWidget *table)
47 {
48     QFileDialog fd;
49     QString fileName;
50     QString destino=this->icca->getDestino();
51     cca_input_format *inf=NULL;
52
```

```
53      if(!destino.isEmpty())
54      {
55          fileName=fd.getOpenFileName(this,tr("Abir Archivo"),
                destino,tr("Audio Wav (*.wav)"));
56          if(!fileName.isEmpty())
57          {
58              if(!fileExists(fileName))
59              {
60                  inf=cca_recognition((char *)fileName.
                        toStdString().c_str());
61                  if(inf!=NULL)
62                  {
63                      int row=table->rowCount();
64                      QLineEdit *ledit;
65                      QSpinBox *sbox=new QSpinBox();
66                      QCheckBox *cbox=new QCheckBox();
67                      sbox->setMaximum(this->icca->getCores());
68                      sbox->setMinimum(1);
69                      cbox->setDisabled(true);
70                      cbox->setChecked(false);
71                      table->insertRow(row);
72                      table->setRowHeight(row,30);
73
74                      table->setCellWidget(row,0,new QLineEdit
                            ());
75                      ledit=(QLineEdit *)table->cellWidget(row
                            ,0);
76                      ledit->setText(fileName);
77                      ledit->setReadOnly(true);
78                      table->setCellWidget(row,1,sbox);
79                      table->setCellWidget(row,2,cbox);
80                      icca->insertFile(fileName,inf);
81                      inf=NULL;
82                      ledit=NULL;
83                      sbox=NULL;
84                      cbox=NULL;
85                  }
86                  else{
87                      QMessageBox::information(this, tr("CCA
                            MP3 encoder"),tr("Formato no soportado
                            o el archivo esta corrupto"));
88                  }
89              }
```

```
90                  else{
91                      QMessageBox::information(this, tr("CCA MP3 encoder"),tr("El Archivo ya se encuentra en la tabla"));
92                  }
93              }
94          }
95          else{
96              QMessageBox::information(this, tr("CCA MP3 encoder"), tr("Primero elija la ruta destino "));
97          }
98      }
99
100     bool MainWindow::fileExists(QString name){
101
102         int rows=this->ui->tbl_audio->rowCount();
103         QLineEdit *ledit;
104         bool ret=false;
105         if(rows>0)
106         {
107             for(int i=0;i<rows && !ret;i++)
108             {
109                 ledit=(QLineEdit *)this->ui->tbl_audio->cellWidget(i,0);
110                 if(name.compare(ledit->text())==0)
111                     ret=true;
112             }
113         }
114         return ret;
115     }
116
117     void MainWindow::on_cmdRemove_clicked()
118     {
119         if(ui->tbl_audio->rowCount()>0)
120         {
121             int row=ui->tbl_audio->currentRow();
122             if(row>=0)
123             {
124                 this->icca->removeFile(row);
125                 ui->tbl_audio->removeRow(row);
126             }
127         }
128     }
```

```
129
130 void MainWindow::on_actionAbout_triggered()
131 {
132     about *a=new about(this);
133     a->setWindowFlags(Qt::Popup);
134     a->show();
135 }
136
137 void MainWindow::on_cmdBuscar_clicked()
138 {
139     QFileDialog fd;
140     QString Directory=fd.getExistingDirectory(this, tr("Abrir
        Directorio"),"/home",QFileDialog::ShowDirsOnly |
        QFileDialog::DontResolveSymlinks);
141     if(!Directory.isEmpty())
142     {
143         this->icca->setDestino(Directory);
144         this->ui->txtDestino->setText(Directory);
145     }
146 }
147
148 void MainWindow::on_cmdClean_clicked()
149 {
150     int rows=this->ui->tbl_audio->rowCount();
151     if(rows>0)
152     {
153         for(int i=0;i<rows;i++)
154             this->ui->tbl_audio->removeRow(0);
155         this->icca->cleanFiles();
156
157         if(this->icca->getWork()==true)
158         {
159             this->icca->setWork(false);
160             this->ui->menuInfo->setEnabled(true);
161             this->ui->cmdAdd->setEnabled(true);
162             this->ui->cmdRemove->setEnabled(true);
163             this->ui->cmdStart->setEnabled(true);
164             this->ui->cmdStop->setEnabled(true);
165             this->ui->actionAdd->setEnabled(true);
166             this->ui->actionRemove->setEnabled(true);
167             this->ui->cmdClean->setEnabled(true);
168             this->ui->actionStart->setEnabled(true);
169             this->ui->actionStop->setEnabled(true);
```

```
170                 this->ui->actionBrowse->setEnabled(true);
171                 this->ui->frameTable->setEnabled(true);
172                 this->ui->frameDestino->setEnabled(true);
173             }
174         }
175 }
176
177 void MainWindow::on_cmdStart_clicked()
178 {
179     QSpinBox *sbox;
180     int *parts;
181     int i,j,n=this->ui->tbl_audio->rowCount();
182
183     if(n>0)
184     {
185         this->ui->menuInfo->setEnabled(false);
186         this->ui->cmdAdd->setEnabled(false);
187         this->ui->cmdRemove->setEnabled(false);
188         this->ui->cmdStart->setEnabled(false);
189         this->ui->cmdClean->setEnabled(false);
190         this->ui->actionClean->setEnabled(false);
191         this->ui->actionAdd->setEnabled(false);
192         this->ui->actionRemove->setEnabled(false);
193         this->ui->actionStart->setEnabled(false);
194         this->ui->actionBrowse->setEnabled(false);
195         this->ui->frameTable->setEnabled(false);
196         this->ui->frameDestino->setEnabled(false);
197
198         parts=new int[n];
199         for(i=0,j=1;i<n;i++)
200         {
201
202             sbox=(QSpinBox *)this->ui->tbl_audio->cellWidget(
                    i,1);
203             j=sbox->value();
204             parts[i]=j;
205             sbox=NULL;
206             j=1;
207         }
208         this->icca->encoder(parts);
209         this->icca->setWork(true);
210     }
211 }
```

```
212
213 void MainWindow::on_cmdStop_clicked()
214 {
215     if(this->icca->getWork()==true)
216         this->on_cmdClean_clicked();
217 }
218
219 void MainWindow::on_actionExit_triggered()
220 {
221     /*salir*/
222       this->icca->cleanFiles();
223       this->on_cmdClean_clicked();
224     /**/
225     this->close();
226 }
227
228 void MainWindow::on_actionAdd_triggered()
229 {
230     this->on_cmdAdd_clicked();
231 }
232
233 void MainWindow::on_actionRemove_triggered()
234 {
235     this->on_cmdRemove_clicked();
236 }
237
238 void MainWindow::on_actionClean_triggered()
239 {
240     this->on_cmdClean_clicked();
241 }
242
243 void MainWindow::on_actionBrowse_triggered()
244 {
245     this->on_cmdBuscar_clicked();
246 }
247
248 void MainWindow::on_actionStart_triggered()
249 {
250     this->on_cmdStart_clicked();
251 }
252
253 void MainWindow::on_actionStop_triggered()
254 {
```

```
255     this->on_cmdStop_clicked();
256 }
257
258 void MainWindow::finishEncode()
259 {
260     this->ui->cmdStop->setEnabled(false);
261     this->ui->actionStop->setEnabled(false);
262     this->ui->cmdClean->setEnabled(true);
263     this->ui->actionClean->setEnabled(true);
264 }
265
266 void MainWindow::checkRow(int i)
267 {
268     QCheckBox *cbox=(QCheckBox *)this->ui->tbl_audio->
          cellWidget(i,2);
269     cbox->setChecked(true);
270 }
```

B.3. about.h

Definición de la clase *about*, la cual representa el cuadro de diálogo **Acerca de**, que contiene información de la aplicación.

.

```
1  #ifndef ABOUT_H
2  #define ABOUT_H
3
4  #include <QDialog>
5
6  namespace Ui {
7      class about;
8  }
9
10 class about : public QDialog
11 {
12     Q_OBJECT
13
14 public:
15     explicit about(QWidget *parent = 0);
16     ~about();
17
18 private:
```

```
19      Ui::about *ui;
20
21  private slots:
22      void on_cmdLicencia_clicked();
23  };
24
25  #endif // ABOUT_H
```

B.4. about.cpp

Funciones de la clase *about*, para el manejo de los eventos de la interfaz.

```
1  #include "about.h"
2  #include "licencia.h"
3  #include "ui_about.h"
4
5  about::about(QWidget *parent) :
6      QDialog(parent),
7      ui(new Ui::about)
8  {
9      ui->setupUi(this);
10 }
11
12 about::~about()
13 {
14     delete ui;
15 }
16
17 void about::on_cmdLicencia_clicked()
18 {
19     licencia *a=new licencia(this);
20     a->setWindowFlags(Qt::Popup);
21     a->show();
22 }
```

B.5. licencia.h

Funciones de la clase *licencia*, la cual representa el cuadro de diálogo que contiene la licencia de la aplicación.

```
1  #ifndef LICENCIA_H
2  #define LICENCIA_H
3
4  #include <QDialog>
5
6  namespace Ui {
7      class licencia;
8  }
9
10 class licencia : public QDialog
11 {
12     Q_OBJECT
13
14 public:
15     explicit licencia(QWidget *parent = 0);
16     ~licencia();
17
18 private:
19     Ui::licencia *ui;
20 };
21
22 #endif // LICENCIA_H
```

B.6. licencia.cpp

Constructor de la clase *licencia*.

```
1  #include "licencia.h"
2  #include "ui_licencia.h"
3
4  licencia::licencia(QWidget *parent) :
5      QDialog(parent),
6      ui(new Ui::licencia)
7  {
8      ui->setupUi(this);
9  }
10
11 licencia::~licencia()
12 {
13     delete ui;
14 }
```

B.7. cca_interface.h

Definición de la clase *cca_interface*, para la interacción entre la GUI y los módulos desarrollados.

```
1  #ifndef CCA_INTERFACE_H
2  #define CCA_INTERFACE_H
3
4  #include <QMainWindow>
5  #include <QString>
6  #include <list>
7  using namespace std;
8
9  #include "cca-threads.h"
10
11 class cca_interface
12 {
13     private:
14         list<thread_file> encodFiles;
15         QString destino;
16         int cores;
17         pthread_t *idThread;
18         bool work;
19         QMainWindow *patern;
20         static void * start(void * arg);
21         int numthreads;
22     public:
23         cca_interface(QMainWindow *p);
24         void encoder(int *parts);
25         int getCores();
26         QString getDestino();
27         void setDestino(QString des);
28         void insertFile(QString path, cca_input_format *inf);
29         void removeFile(int item);
30         void cleanFiles();
31         bool getWork();
32         void setWork(bool w);
33         int getNumFiles();
34         thread_file getFile();
35         void checkRow(int i);
36         void finishEncode();
37 };
38 #endif // CCA_INTERFACE_H
```

B.8. cca_interface.cpp

Funciones de la clase *cca_interface*, para la interacción entre la GUI y los módulos desarrollados.

```
1  #include <iostream>
2  using namespace std;
3  #include <QObject>
4  #include <QFileInfo>
5
6  #include "cca_interface.h"
7  #include "mainwindow.h"
8
9  cca_interface::cca_interface(QMainWindow *p){
10     this->patern=p;
11     this->destino="";
12     this->work=false;
13     this->cores=cca_get_num_cores();
14     if(this->cores<=0)
15         this->cores=1;
16     this->numthreads=0;
17     this->idThread=NULL;
18 }
19
20 void cca_interface::setWork(bool w){
21         this->work=w;
22 }
23
24 bool cca_interface::getWork(){
25     return this->work;
26 }
27
28 int cca_interface::getCores(){
29     return this->cores;
30 }
31
32 QString cca_interface::getDestino(){
33     return this->destino;
34 }
35
36 void cca_interface::setDestino(QString des){
37     this->destino=des;
38 }
39
```

```
40 void cca_interface::insertFile(QString path, cca_input_format
       *inf)
41 {
42     thread_file argFile;
43     QFileInfo finfo(path);
44     QString fname;
45     QString outPath;
46     fname=finfo.fileName();
47     fname=fname.replace(fname.length()-3,3,QObject::tr("mp3")
           );
48     outPath=this->destino+fname;
49     strncpy(argFile.inPath, path.toStdString().c_str(), path.
           length()+1);
50     strncpy(argFile.outPath, outPath.toStdString().c_str(),
           outPath.length()+1);
51     argFile.inf=inf;
52     argFile.partition=1;
53     this->encodFiles.push_back(argFile);
54 }
55
56 void cca_interface::removeFile(int item){
57     list<thread_file>::iterator it;
58     it=this->encodFiles.begin();
59     for(int i=0;i<item;i++,it++){}
60     this->encodFiles.erase(it);
61 }
62
63 thread_file cca_interface::getFile(){
64     thread_file ret=this->encodFiles.front();
65     this->encodFiles.pop_front();
66     return ret;
67 }
68
69 void cca_interface::cleanFiles(){
70     this->encodFiles.clear();
71 }
72
73 int cca_interface::getNumFiles()
74 {
75     return this->encodFiles.size();
76 }
77
78 void cca_interface::checkRow(int i)
```

```
79 {
80     MainWindow *mw=(MainWindow *)this->patern;
81     mw->checkRow(i);
82 }
83
84 void cca_interface::finishEncode()
85 {
86     MainWindow *mw=(MainWindow *)this->patern;
87     mw->finishEncode();
88 }
89
90 void cca_interface::encoder(int *parts){
91
92     list<thread_file>::iterator it;
93     int i, n=this->encodFiles.size();
94
95     it=this->encodFiles.begin();
96     for(i=0;i<n;i++,it++)
97     {
98         it->partition=parts[i];
99     }
100    if(this->idThread!=NULL)
101        delete this->idThread;
102    this->idThread=new pthread_t;
103    pthread_create(this->idThread, NULL, cca_interface::start
       , (void *)(this));
104 }
105
106 /** Hilo que inicia la codificacion de archivos y espera a
    que terminen*/
107 void * cca_interface::start(void *arg){
108     int i,j,n,cores,*ac,a;
109     thread_file * targ;
110     pthread_t *idThreads;
111     cca_interface * args=(cca_interface *)arg;
112
113     cores=args->getCores();
114     n=args->getNumFiles();
115     idThreads = new pthread_t[n];
116     ac= new int[n];
117
118     for(i=0,a=0;i<n;i++)
119     {
```

```
120         targ=new thread_file;
121         *targ=args->getFile();
122         ac[i]=targ->partition;
123         /*Si los nucleos estan cupados espera a que se
               liberen para su codificacion*/
124         if(a==cores || (cores-a)<ac[i])
125         {
126             for(j=0;j<n && (cores-a)<ac[i];j++)
127             {
128                 if(ac[j]!=0)
129                 {
130                     pthread_join(idThreads[j],NULL);
131                     a=a-ac[j];
132                     ac[j]=0;
133                     args->checkRow(j);
134                 }
135             }
136
137         }
138         a+=ac[i];
139         pthread_create(&(idThreads[i]), NULL,
               thread_function_file, (void *)targ);
140         targ=NULL;
141     }
142
143     /*Espera a que todos los hilos terminen*/
144     for(j=0;j<n;j++)
145     {
146         if(ac[j]!=0)
147         {
148             pthread_join(idThreads[j],NULL);
149             args->checkRow(j);
150         }
151     }
152     delete idThreads;
153     delete ac;
154     args->finishEncode();
155     pthread_exit(NULL);
156 }
```

Apéndice C

Instalación

A continuación se muestra los pasos a seguir para la instalación. La aplicación se encuentra empaquetada en el paquete cca-1.1.tar. El contenido del paquete es el siguiente:

```
drwxr-xr-x  cca-qt // Fuentes de la GUI
-rw-r--r--  COPYING // Copyright
-rw-r--r--  INSTALL // Instrucciones de instalación
lrwxrwxrwx  lame -> lame-398-2/ // Liga a las fuentes de Lame-3.98.2
drwxr-xr-x  lame-398-2 // Fuentes de LAME-3.98.2
drwxr-xr-x  libcca // Fuentes de la biblioteca desarrollada (libcca.a)
-rw-r--r--  Makefile // Makefile para la instalación
-rw-r--r--  README // Pequeña descripción de la Aplicación
```

C.1. Requerimientos

- Compiladores **gcc** y **g++**, versión >= 4.3.2.
- Biblioteca **libpthread** y archivos de desarrollo **libpthread-dev**, para el manejo de hilos.
- Bibliotecas, archivos de desarrollo y herramientas de construcción de Qt[5] versión >= 4.4.3.
- Herramienta de contrucción **make** versión >= 3.81.
- Utilidad **tar** versión >= 1.20.
- Copia del código de LAME[7], versión >= 3.98.2.

C.2. Instalación

La instalación se lleva a cabo desde consola. El paquete cca-1.1.tar contiene la versión 3.98.2 del código fuente de LAME, a contuniación se mostraran dos formas de instalar la aplicación suponiendo que el paquete cca-1.1.tar se encuentra en */home/angel/programas*.

Instalación a partir de la versión integrada de LAME

```
$ cd /home/angel/programas
$ tar -xvvf cca-1.1
$ cd cca-1.1/
$ make

//Obtener permisos de super usuario(root)
$ su
# make install
# exit

//Ejecutar
$ cca-qt
```

Instalación a partir de una versión distinta de LAME

Si se desea utilizar una versión distinta de LAME, se puede obtener en `http://lame.sourceforge.net/index.php`. Una vez descargada la nueva versión se deben seguir los siguientes pasos, suponiendo que se descargo la versión 4.03 de LAME(lame-4.03.tar.gz) y se guardo en */home/angel/Descargas*:

```
$ cd /home/angel/Descargas
$ tar -xvvzf lame-4.03.tar.gz
$ cd /home/angel/programas
$ tar -xvvf cca-1.1
$ cd cca-1.1/
& ls -l
.
.
lrwxrwxrwx  lame -> lame-398-2/ // liga anterior
.
.

//Se crea una liga a las fuentes nuevas
```

```
$ rm ./lame
$ ln -s /home/angel/Descargas/lame-4.03/ ./lame
& ls -l
.
.
lrwxrwxrwx  lame -> /home/angel/Descargas/lame-4.03/ // liga nueva
.
.
$ make

//Obtener permisos de super usuario(root)
$ su
# make install
# exit

//Ejecutar
$ cca-qt
```

C.3. Desinstalación

Para la desinstalación haga lo siguiente desde consola.

```
$ cd /home/angel/programas
$ cd cca-1.1/

//Obtener permisos de super usuario(root)
$ su
# make uninstall
# exit
```

Bibliografía

[1] Pohlmann, Ken C. "Principios de audio digital". Traducido por Antonio Míguez Olivares. Madrid: McGraw Hill, 2002, 724 p., ISBN: 84-481-3625-X

[2] Stallings, W. "*Operating systems: internals and design principles*", 4th. Ed, Prentice Hall, 2001.

[3] Tanenbaum, A. S., van Steen M., "*Distributed Systems: Principles and Paradigms*", Prentice Hall, 2002.

[4] GCC, The GNU Compiler Collection.
`http://gcc.gnu.org/`. Consultada en agosto de 2010.

[5] Qt: A cross-platform application and UI framework .
`http://qt.nokia.com/products`. Consultada en agosto de 2010.

[6] FFmpeg: programa que convierte muestras de audio y video.
`http://ffmpeg.org/`. Consultada en agosto de 2010.

[7] Lame: codificador de MP3.
`http://lame.sourceforge.net/index.php`. Consultada en agosto de 2010.

[8] Perl Audio Converter: herramienta para la conversión de varios tipos de audio de un formato a otro.
`http://pacpl.sourceforge.net/`. Consultada en agosto de 2010.

[9] Switch Audio Converter Software: aplicación para la conversión de varios tipos de audio de un formato a otro.
`http://www.nch.com.au/switch/index.html`. Consultada en agosto de 2010.

[10] MP3 Converter: aplicación para la conversión de varios tipos de audio de un formato a otro.
`http://www.wav-mp3.com/mp3-converter.htm`. Consultada en agosto de 2010.

[11] Xilisoft Audio Converter: aplicación para la conversión de varios tipos de audio de un formato a otro.
`http://www.xilisoft.com/audio-converter.html`. Consultada en agosto de 2010.

[12] WAVE: Información sobre el formato WAV.
https://ccrma.stanford.edu/courses/422/projects/WaveFormat/. Consultada en agosto de 2010.

[13] WAVE: Información sobre el formato WAV.
http://es.wikipedia.org/wiki/Waveform_Audio_Format. Consultada en agosto de 2010.

[14] MP3: Información sobre el formato MP3.
http://www.iis.fraunhofer.de/EN/bf/amm/products/mp3/index.jsp. Consultada en agosto de 2010.

[15] MPEG: Grupo de Expertos en Imágenes en Movimiento.
http://www.mpeg.org/ Consultada en agosto de 2010.

[16] ISO: Organización Internacional para la Estandarización.
http://www.iso.org/ Consultada en agosto de 2010.

[17] IEC: Comisión Electrotécnica Internacional.
http://www.iec.ch/ Consultada en agosto de 2010.

[18] ITU: Unión Internacional de Telecomunicaciones.
http://www.itu.int/es/pages/default.aspx Consultada en agosto de 2010.

[19] Fraunhofer ISS: Instituto Fraunhofer.
http://www.iis.fraunhofer.de/EN/bf/amm/ Consultada en agosto de 2010.

[20] Thomson Multimedia.
http://www.thomson.net/ Consultada en agosto de 2010.

[21] The Pthreads API.
https://computing.llnl.gov/tutorials/pthreads/ Consultada en agosto de 2010.

[22] Descripción del formato MP3.
http://www.mp3-tech.org/ Consultada en agosto de 2010

[23] GNU General Public Licenses.
http://www.gnu.org/copyleft/gpl.html. Consultada en agosto de 2010.

[24] Audacity: Editor de Audio.
http://audacity.sourceforge.net. Consultada en agosto de 2010.

[25] k3b: Utilidad de grabación.
http://k3b.plainblack.com/. Consultada en agosto de 2010.

[26] Arson: Utilidad de grabación.
http://arson.sourceforge.net/index.php. Consultada en agosto de 2010.

[27] iTunes-LAME: Codificador MP3.
`http://blacktree.com/?itunes-lame`. Consultada en agosto de 2010.

[28] SecondSpin: Codificador MPEG.
`http://www.helsinki.fi/~lakahone/amiga/secondspin/`. Consultada en agosto de 2010.

[29] WINAMP: Reproductor de Medios.
`http://www.winamp.com/`. Consultada en agosto de 2010.

[30] Tracktion3: Editor de Audio.
`http://www.mackie.com/products/tracktion3/`. Consultada en agosto de 2010.

[31] Acoustica: Editor de Audio.
`http://www.acondigital.com/us_Acoustica1.html`. Consultada en agosto de 2010.

[32] Audion: Reproductor y codificador de mp3.
`http://www.panic.com/audion/`. Consultada en agosto de 2010.

[33] CD Copy: Grabación y copia de audio.
`http://www.cdcopy.sk/`. Consultada en agosto de 2010.

[34] POSIX: *Portable Operating System Interface UNIX*.
`http://standards.ieee.org/regauth/posix/`. Consultada en agosto de 2010.

[35] Mplayer: The Movie Player.
`http://www.mplayerhq.hu/design7/news.html`. Consultada en agosto de 2010.

[36] tooLame: Codificador MP2.
`http://toolame.sourceforge.net/`. Consultada en agosto de 2010.