Escuela Politécnica Superior de Linares

Máster en Ing. De Telecomunicaciones

UNIVERSIDAD DE JAÉN
ESCUELA POLITÉCNICA SUPERIOR DE LINARES

Trabajo Fin de Máster

AUTOMATIC ACCOMPANIMENT USING SPECTRAL DESCOMPOSITION AND ONLINE DYNAMIC TIME WARPING

Alumno: **Pedro José Pérez Marín**

Tutores: Pedro Vera Candeas
Francisco J. Rodríguez Serrano
Depto.: Ingeniería de Telecomunicación

Junio, 2016

ÍNDICE DE CONTENIDOS

1. RESUMEN 4

2. INTRODUCCIÓN 5

2.1 INTRODUCCIÓN A LA ESPECIFICACIÓN MIDI 5

2.1.1. ¿Qué es el MIDI? 5

2.1.2. Los canales MIDI 5

2.1.3. Bytes de estados y de datos 6

2.1.4. Estructura de un mensaje 6

2.1.5. Tipos de mensajes 6

2.1.6. Mensajes de canal 7

2.1.6.1. Note On *7*

2.1.6.2. Note Off *8*

2.1.6.3. Polyphonic Aftertouch (postpulsación polifónica) *8*

2.1.6.4. Channel Aftertouch (postpulsación de canal) *8*

2.1.6.5. Pitch Bend (variación de la altura) *9*

2.1.6.6. Program Change (cambio de programa) *9*

2.1.6.7. Los mensajes de control change (cambio de control) *12*

2.1.7. Mensajes de sistema *14*

2.1.7.1. Mensajes comunes de sistema *14*

2.1.7.2. Mensajes de sistema de tiempo real *14*

2.1.7.3. El sistema exclusivo *14*

2.2 ESTUDIO PREVIO Y MATERIAL UTILIZADO 15

2.2.2. Herramientas utilizadas en la creación del trabajo 15

2.2.3. Creación del programa 16

3. OBJETIVOS 16

4. MATERIALES Y MÉTODOS 17

4.1 ESTADO DEL ARTE 17

4.1.1. Sintetizador 17

4.1.2. Historia y evolución 18

4.1.3. Fluidsynth 21
4.1.4. Mensajes OSC 25
4.2 PROGRAMA DESARROLLADO 28
4.2.1. Biblioteca MidiFile 29
4.2.2. Cambio de formato 32
4.2.3. Hebras 33
4.2.3.1. Hebra principal 33
4.2.3.2. Hebra Reproducir 35
4.2.3.3. Hebra tempo 39
4.2.4. Sincronización de hebras 40
4.2.5. Diagrama con la información del programa 41
4.2.6. Interrupciones y dormir 42
5. Resultados y discusión 43
5.1 PRUEBAS 43
5.2 TRABAJO FINAL OBTENIDO 45
5.3 PROBLEMAS ENCONTRADOS 45
6. CONCLUSIONES 46
7. ANEXOS 47
7.1 ANEXO I. MANUAL DE USUARIO 47
7.2 ANEXO II. DOCUMENTACIÓN TÉCNICA 50
7.3 ANEXO III. ÍNDICE DE FIGURAS 74
7.4 ANEXO IV. ÍNDICE DE TABLAS 75
8. REFERENCIAS BIBLIOGRÁFICAS 75

1 RESUMEN

Español

El presente trabajo fin de máster consiste en desarrollar un instrumento virtual en C que sea capaz de procesar las notas de un archivo de audio MIDI para ser enviadas a un sintetizador, el cual se encargará de reproducirlas en función de un determinado instrumento indicado en cada archivo de audio.

El software permite recibir distintos eventos en tiempo real a través de mensajes OSC. Estos son, el cambio de TEMPO, para modificar la velocidad de ejecución de la pieza musical, el mensaje STOP para detener la reproducción y el mensaje PLAY para continuar con la reproducción desde donde se requiera, ya sea desde el comienzo o en cualquier otro instante.

Este trabajo forma parte de un proyecto llevado acabo por algunos de los componentes del departamento de ingeniería de telecomunicación de la escuela politécnica superior de Linares, el cual consiste en desarrollar una aplicación para Smartphone, tabletas y ordenadores que sea capaz de realizar un acompañamiento musical en tiempo real a un músico que esté interpretando una determinada obra.

Existen semejantes sistemas de acompañamiento musical como el denominado antescofo, cuyo objetivo no es otro que sincronizar la actuación en directo de uno o varios músicos con su respectiva pieza musical. En la referencia bibliográfica número [27] se puede visualizar un video en el que se muestra el funcionamiento de este. También se puede obtener más información acerca de sistemas de acompañamiento musical en la referencia bibliográfica [26].

A lo largo de esta memoria se verá una introducción a la especificación MIDI, para entender que es y cómo funciona, el estudio previo realizado, historia y evolución del sintetizador, el desarrollo en sí del programa con las pruebas realizadas y problemas encontrados así como las conclusiones obtenidas finalmente.

Inglés

This Master Thesis is developed to develop a virtual instrument in C that is capable of processing the notes from a MIDI audio file to be sent to a synthesizer which will play them in terms of a determined instrument indicated in MIDI channel.

The software allows to receive different events in real time via OSC messages. These are, tempo change, to change the execution speed of the track, the stop message to stop the playback and the play message to resume playback from where required, either from the start or at any time.

This work is part of a project made by some people in the telecommunications engineering department of the EPSL (Jaen University), which is to develop a mobile, tablet or computer app, it should do a musical accompaniment in real time to a musician who is playing a determined musical piece.

There are musical accompaniment systems like the called *antescofo*, whose objective is to synchronize the live performance of some musician with the MIDI audio file. You can find a video in the bibliography (number 27) in which appear how it works. Also you can get more information about musical accompaniment systems in the bibliography (number 26).

In this memory you will see an introduction to the MIDI specification, to understand what this is and how this works, previous study, history and evolution of synthesizer, how the software has been developed, tests done and problems found.

2 INTRODUCCIÓN

2.1 INTRODUCCIÓN A LA ESPECIFICACIÓN MIDI

La información MIDI mostrada a continuación ha sido obtenida de una guía de audio digital y MIDI, cuyo enlace se puede obtener en la referencia bibliográfica número [4].

2.1.1. ¿Qué es el MIDI?

MIDI es el acrónimo de Musical Instruments Digital Interface. No se trata de una forma de comprensión de audio digital, sino de un protocolo digital de comunicaciones, surgido del entendimiento entre fabricantes de equipos musicales electrónicos, permitiendo la comunicación entre ellos y la comunicación con los ordenadores. Está basado en los sistemas binarios y hexadecimales.

Existen confusiones respecto al MIDI como pensar que es algo material o un formato de audio en sí mismo, pero realmente, se trata de un conjunto de comandos o mensajes digitales que circulan entre dispositivos MIDI dando órdenes a los mismos respecto a lo que deben hacer. De modo que lo que suena en realidad son los dispositivos en función de las órdenes dadas por los archivos MIDI.

2.1.2. Los canales MIDI

El protocolo MIDI permite que los mensajes se envíen a través de dieciséis canales diferentes. Estos canales no corresponden a conexiones físicas separadas, ya que comparten un único cable, sino más bien a direcciones lógicas. Un dispositivo controlador suele enviar por un único canal a la vez, mientras que un dispositivo receptor (un sintetizador) puede ser configurado para recibir en uno o varios canales simultáneos.

Se puede pensar en un canal como en un instrumento virtual independiente. Actualmente, muchos sintetizadores y la totalidad de las tarjetas de sonido son capaces de reproducir varios instrumentos diferentes de forma simultánea. Para que esto sea posible, deben ser capaces de recibir en varios canales MIDI. Un sintetizador que pueda generar cuatro instrumentos simultáneos, deberá como mínimo poder recibir en cuatro canales MIDI diferentes.

2.1.3. Bytes de estados y de datos

Los mensajes MIDI están formados por dos o tres bytes. Estos bytes se dividen en dos categorías, bytes de estado y bytes de datos, de acuerdo con el valor de su bit más significativo que en los bytes de estado vale 1 mientras que en los de datos vale 0. Los siete bits libres restantes son los que condicionan el número de posibles programas en el mensaje. De este modo, los datos MIDI están siempre comprendidos entre los valores decimales 0 y 127, binarios 00000000 y 01111111. En el siguiente apartado se detalla la construcción los dos tipos de bytes existentes.

2.1.4. Estructura de un mensaje

Cualquier mensaje MIDI está compuesto por un primer byte de estado, el cual determina el tipo del mensaje, y uno o dos bytes restantes de datos dependiendo del tipo de mensaje.

En el byte de estado, solo tres de los siete bits disponibles son los que determinan el tipo de mensaje, sin olvidar que el bit más significativo está siempre a 1. Los cuatro restantes indican el canal al que el mensaje va dirigido, de ahí el motivo de la posibilidad de trabajar con 16 canales MIDI (2^4).

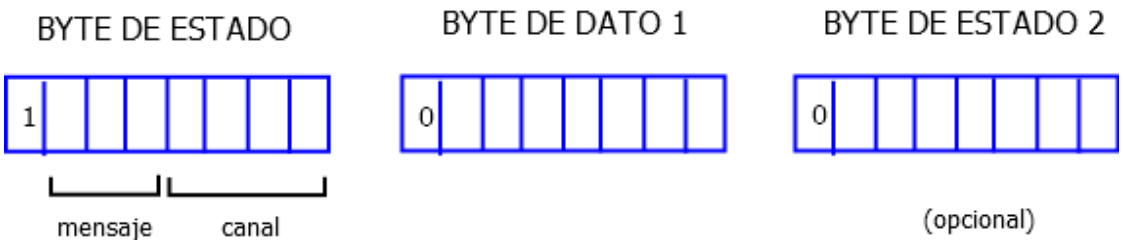

Figura 1. Estructura binaria de un mensaje MIDI

2.1.5. Tipos de mensajes

Según lo explicado en el apartado anterior, se deduce que existen ocho (2^3) tipos de mensajes MIDI diferentes.

Nombre	Binario	Hexadecimal	Dato 1	Dato 2
Note Off	1000 nnnn	8 N	Altura	Velocidad
Note On	1001 nnnn	9 N	Altura	Velocidad
Poly. Aftertouch	1010 nnnn	A N	Altura	Presión
Control Change	1011 nnnn	B N	Tipo de control	Intensidad
Chan. Aftertouch	1100 nnnn	C N	Presión	
Pitch Bend	1101 nnnn	D N	MSByte	LSByte
Program Change	1110 nnnn	E N	programa	
System Message	1111 xxxx	F X		

Tabla 1. Cuadro sinóptico de los mensajes MIDI

- nnnn: son los cuatro bits que indican el canal al que va dirigido el mensaje, de manera que 0000 es el canal 1 y 1111 es el canal 16.
- N: se corresponde con el carácter hexadecimal de este canal (0-F).
- Todos los bytes de datos presentan una resolución de siete bits, con valores decimales comprendidos entre o y 127.

  Cuando en la tabla el segundo bytes de datos está en blanco, significa que el mensaje utiliza un único byte de datos, como ocurre con el Channel Aftertouch y el Program Change.
- En el mensaje Pitch Bend, los dos bytes de datos se combinan para formar un único valor con catorce bits de resolución, comprendidos entre -8192 y +8191.
- Los mensajes de sistema no se aplican a un canal específico.7

Cabe destacar que no todos los dispositivos MIDI tienen porque entender todos los mensajes posibles. En ese caso, cuando un dispositivo desconoce un determinado mensaje, lo envía como cualquier otro evento pero ignorando su significado.

2.1.6. Mensajes de canal

Son aquellos que se aplican a un único canal, el cual será determinado por los cuatro bits menos significativos del byte de estado, como se ha comentado anteriormente. Este tipo de mensaje comprenden la gran mayoría de mensajes MIDI.

2.1.6.1. Note On

Este tipo de mensaje indica el inicio de una determinada nota, por ejemplo cuando se pulsa la tecla de un teclado.

- El primer byte de datos indica la altura de la nota. Como se dispone de siete bits, se deduce que el MIDI comprende 128 notas posibles, siendo 0 la nota más grave y 127 la nota más aguda. Teniendo en cuenta que existen doce notas por octava, el MIDI tiene una tesitura de más de diez octavas que se corresponden con el número de octavas que el oído humano es capaz de captar.
- El segundo byte indica la velocidad de ataque, es decir, la intensidad sonora con la que una nota es reproducida. Por ejemplo, en un piano, la velocidad de ataque viene determinada por la fuerza con la que se aprieta una tecla.

Algunos dispositivos no detectan la velocidad de ataque, por lo que se establece una velocidad constante 64. Si dicha velocidad obtiene un valor de 0, la nota correspondiente será silenciada.

A continuación se muestra una tabla con todas las notas MIDI en función de su octava correspondiente:

Octava	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

Tabla 2. Tabla de notas MIDI

2.1.6.2. Note Off

Este mensaje presenta un funcionamiento muy similar al de Note On, indicando la liberación de una determinada nota.

- El primer byte de datos se corresponde con la altura de la nota.
- El segundo byte de datos indica la velocidad de liberación.

Es un mensaje muy poco utilizado ya que la mayoría de dispositivos no responden a este. Por lo que en vez de liberar una nota, se envía un Note On con velocidad 0, ya que todos los sintetizadores entienden dicho mensaje.

2.1.6.3. Polyphonic Aftertouch (postpulsación polifónica)

Algunos teclados de alta gama son capaces de detectar los cambios de presión ejercidos sobre cada una de las teclas, de manera permanente. De este modo, cada vez que se produzca un cambio de presión envían este mensaje.

- El primer byte indica la altura de la nota.
- El segundo byte indica la presión ejercida sobre esa nota.

Aquellos dispositivos capaces de realizar esta operación, suelen tener una opción para desactivarla, ya que el flujo de bytes enviados (varias decenas de mensajes por segundo, para cada nota que se mantenga pulsada) puede provocar atascos.

2.1.6.4. Channel Aftertouch (postpulsación de canal)

Este mensaje es una versión económica del Polyphonic Aftertouch, ya que en vez de enviar

un mensaje de presión por cada nota, lo envía por cada canal aunque en este se mantengan pulsadas diez notas.

- El único byte de datos utilizados indica la presión.

2.1.6.5. Pitch Bend (variación de la altura)

La mayoría de los teclados disponen de unas ruedas giratorias de las cuales, una de ellas se utiliza para desafinar ligeramente el sonido. Cuando la rueda gira, el teclado envía estos mensajes de forman continua.

- Los bytes de datos se combinan para dar un único valor con 14 bits de resolución, proporcional al ángulo de rotación, y comprendido entre -8192 y +8191. Este valor es nulo cuando la rueda está en su posición central.

2.1.6.6. Program Change (cambio de programa)

En MIDI se utiliza el término programa para designar los diferentes tipos de sonidos disponibles en un sintetizador (instrumentos, efectos sonoros... etc). Este tipo de mensaje modifica el programa actual pudiendo ser enviado desde los botones de un teclado o desde el propio ordenador, editándolo en el secuenciador.

- El único byte utilizado indica el número de programa.

A continuación aparece la lista de instrumentos GM (Genaral MIDI).

PIANO	PERCUSIÓN CROMÁTICA	ÓRGANO
1 - Piano de cola	09 – Celesta	17 – Órgano Hammond
2 - Piano brillante	10 – Glockenspiel	18 – Órgano percusivo
3 – Piano eléctrico de cola	11 – Caja de música	19 – Órgano rock
4 – Piano Honky-Tonk	12 – Vibráfono	20 – Órgano de iglesia
5 – Piano eléctrico 1	13 – Marimba	21 – Harmonium
6 – Piano eléctrico 2	14 – Xilófono	22 – Acordeón
7 – Clave	15 – Tubular Bells	23 – Armónica
8 – Clavinet	16 – Dulcimer	24 – Bandoneón

GUITARRA	BAJO	CUERDAS
25 – Guitarra española 26 – Guitarra acústica 27 – Guitarra eléctrica (jazz) 28 – Guitarra eléctrica 29 – Guitarra eléctrica muteada 30 – Guitarra saturada 31 – Guitarra con distorsión 32 – Armónicos guitarra	33 – Bajo acústico 34 – Bajo eléctrico 35 – Bajo eléctrico (púa) 36 – Bajo sin trastes 37 – Bajo Slap 1 38 – Bajo Slap 2 39 – Bajo sintetizado 1 40 – Bajo sintetizado 2	41 – Violín 42 – Viola 43 – Violonchelo 44 – Contrabajo 45 – Tremolo 46 – Pizzicato 47 – Arpa 48 – Timbales
Lista de instrumentos GM (General MIDI)		

ESEMBLE	VIENTO –METALES	VIENTO-MADERA (LENGUETA)
49 – Cuerdas 1 50 – Cuerdas 2 51 – Cuerdas sintetizadas 1 52 – Cuerdas sintetizadas 2 53 – Coro Aahs 54 – Voces Oohs 55 – Voz sintetizada 56 – Golpe de orquesta	57 – Trompeta 58 – Trombón 59 – Tuba 60 – Trompeta con sordina 61 – Trompa 62 – Sección Metales 63 – Metales sintetizados 1 64 – Metales sintetizados 2	65 – Saxofón Soprano 66 – Saxofón Contralto 67 – Saxofón Tenor 68 – Saxofón Barítono 69 – Oboe 70 – Corno Inglés 71 – Fagot 72 – Clarinete
Lista de instrumentos GM (General MIDI)		

VIENTO-MADERA (BISEL)	SINTETIZADOR (DE SOLO)	SINTETIZADOR (DE FONDO)
73 – Flautín	81 – Onda cuadrada	89 – Fantasía / New Age
74 – Flauta travesera	82 – Diente de sierra	90 – Cálido
75 – Flauta de pico	83 – Órgano de vapor	91 – Poli-sintetizador
76 – Flauta de pan	84 – Chiffer	92 – Voz
77 – Botella	85 – Charango	93 – Vidrio de arco
78 – Skakuhachi	86 – Voz	94 – Metálico
79 – Whistle	87 – Quintas	95 – Halo
80 – Ocarina	88 – Bass & Lead	96 – “Sweep”
Lista de instrumentos GM (General MIDI)		

EFECTOS	ÉTNICO	PERCUSIÓN
97 – Lluvia	105 – Sitar	113 – Campanillas
98 – Banda sonora	106 – Banjo	114 – Agogó
99 – Cristal	107 – Shamishen	115 – Caja Metal
100 – Atmósfera	108 – Koto	116 – Caja Madera
101 – Brillo	109 – Kalimba	117 – Taiko
102 – Duendes	110 – Gaita	118 – Timbal
103 – Ecos	111 – Violín celta	119 – Caja sintetizada
104 – Star theme	112 – Shanai	120 – Plato (invertido)
Lista de instrumentos GM (General MIDI)		

SONIDOS
121 – Traste de guitarra
122 – Respiración
123 – Playa
124 – Pájaro
125 – Teléfono
126 – Helicóptero
127 – Aplauso
128 – Disparo

Tabla 3. Lista de instrumentos GM (General MIDI)

2.1.6.7. Los mensajes de control change (cambio de control)

Este mensaje engloba 128 posibles mensajes de control diferentes afectando cada uno de ellos de alguna forma en la calidad del sonido. Se estructura de la siguiente forma:

- El primer byte indica el tipo de control. De los 128 posibles, solo una pequeña parte está asignada, de manera que todavía quedan muchos por definir en un futuro.
- El segundo byte indica el valor de este control utilizando la escala del 0 al 127.

Existen diferentes formas de enviar este tipo de mensajes. Se puede realizar a través de las ruedas que presentan los teclados. También existen paneles de control MIDI, con varios botones o potenciómetros configurables, de forma que el usuario puede decidir el canal y el tipo de control al que asigna cada potenciómetro. O bien a través del propio software secuenciador, siendo esta la alternativa más flexible.

A continuación se detallan los controles más utilizados.

- Control Change 0: Cambio de banco

Si el sintetizador presenta varios bancos de sonido, mediante este control se puede acceder a todos ellos a través del tercer byte de datos. En algunos sintetizadores el cambio de banco se consigue con el *Control 32.*

- Control Change 1: Modulación

Este control se envía a través de una de las ruedas del teclado. El efecto obtenido puede variar de un sintetizador a otro y es programable por el usuario la mayoría de las veces.

- Control Change 7: Volumen

Este control es uno de los más utilizados. A diferencia del Note On que afecta a la intensidad de una sola nota, el control 7 modifica el volumen de todo el canal en general, como si fuera un mezclador. Cuando se envía un *Control 7* con valor 0, el canal dejará de oírse aunque se sigan enviado notas, hasta que se envíe un nuevo *Control 7* con un valor no nulo.

- Control Change 10: Panorama

Este control permite definir la posición sonora de un canal, en un ámbito de 180 grados. Con un valor de 0, la fuente sonora se sitúa a la izquierda, con un valor de 127, la fuente se sitúa a la derecha y permanece en el centro cuando obtiene un valor de 64. También puede ser enviado cualquier valor comprendido entre 0 y 127.

- Control Change 11: Expresión

Este control está pensado para trabajar en conjunto con el *Control 7* (volumen). Cuando la expresión vale 127, el volumen general del canal viene determinado por el valor del Control 7, pero a medida que el valor de la expresión desciende, también lo hace el volumen general del canal.

Esto permite utilizar el Control 7 para controlar la mezcla global y el Control 11 para controlar la articulación particular de algunos fragmentos, teniendo muy en cuenta que, si no se utiliza, el valor del Control 11 debería permanecer a 127.

- Control Change 64: Sostenido

Este control es muy parecido al funcionamiento del pedal del sostenido de un piano. Presenta solo dos posiciones, apagado (0-63) o encendido (64-127). Estando activado, las notas se mantienen más tiempo.

- Control Change 91: Reverberación

Este control indica la proporción entre el sonido directo y el sonido reflejado. Es utilizado para realizar simulaciones acústicas de las salas de conciertos. Cuanto mayor sea una sala, mayor será la reverberación percibida, al igual que cuanto más distante esté una fuente sonora, mayor será la reverberación. De este modo, cuanto mayor sea el valor de este control, más distante parecerá la fuente sonora.

- Control Change 93: Chorus

Este control es utilizado en las grabaciones en estudios. Produce un efecto parecido al que se

obtiene duplicando los instrumentos, por ello, cuanto mayor sea el valor de este control, mayor parecerá el acompañamiento.

2.1.7. Mensajes de sistema

Este grupo están formados por todos los mensajes cuyo byte de estado comienza con 1111. Presentan un comportamiento distinto a los mensajes de canal, ya que como bien su nombre indica, afectan globalmente al comportamiento de los dispositivos que los reciba. Por tanto, los cuatro bits restantes no indican el número de canal. De hecho, estos cuatro bits definen dieciséis mensajes diferentes que se pueden repartir en tres grupos: los mensajes comunes de sistema, los mensajes de sistema de tiempo real y los mensajes de sistema exclusivo.

2.1.7.1. Mensajes comunes de sistema

Estos mensajes suelen ir destinados a secuenciadores, pudiendo ser enviados desde otro secuenciador u otro dispositivo dotado del interfaz adecuado. Permite posicionar automáticamente un secuenciador en un determinado fragmento de una determinada pieza, desde el dispositivo de control.

El más importante es el *MIDI Time Code* (MTC) que consta de dos bytes (siendo 1111 0001 su byte de estado) y se utiliza para sincronizar secuenciadores con grabadoras multipistas o magnetoscopios de vídeo.

2.1.7.2. Mensajes de sistema de tiempo real

Estos mensajes son utilizados para coordinar y sincronizar dispositivos MIDI que normalmente funcionarían de forma independiente, como dos secuenciadores. En este caso, uno funcionará como maestro y el otro como esclavo. Aunque los secuenciadores trabajan por defecto con un reloj interno (maestros), siempre es posible configurarlos para que dependan de un reloj externo (esclavos).

2.1.7.3. El sistema exclusivo

Todos los dispositivos MIDI poseen algunas características peculiares, condicionadas por su hardware y software interno, y difíciles de incluir en un estándar interno. Por ello, los fabricantes acordaron dejar un grupo de mensajes de formato libre, para uso particular de cada dispositivo.

Estos mensajes de sistema exclusivo incluyen después del byte de estado un byte (que en este caso siempre es 1111 0000) con un código propio del fabricante y otro específico del modelo. A continuación el mensaje puede tener cualquier longitud, por lo que para indicar el fin del mensaje, se incluye el byte de estado EOX (End of Exclusive) que vale 1111 0111. A continuación se puede apreciar la estructura de un mensaje de sistema exclusivo.

INI. SysEx	Id. Fabricante	Id. Modelo	Cuerpo del mensaje	EOX
1111 0000	0nnn nnnn	0nnn nnnn	Cualquier número de bytes	1111 0111

Tabla 4. Mensaje de sistema exclusivo

2.2 ESTUDIO PREVIO Y MATERIAL UTILIZADO

2.2.1. Búsqueda de información e investigación

Al comienzo de este trabajo fin de máster se realizó una amplia búsqueda de información variada, perteneciente a distintas áreas, para su posterior desarrollo. En primer lugar se indagó sobre la especificación MIDI en diferentes páginas de internet, cuya dirección aparece indicada en las referencias bibliográficas [4] y [17], con el objetivo de comprender su estructura y funcionamiento para realizar una programación lo más correcta posible.

En segundo lugar, se investigó sobre qué eran los sintetizadores y como trabajan, ya que es uno de los elementos clave en este proyecto al ser el encargado de reproducir las notas MIDI. Además, se realizó una comparativa entre diferentes sintetizadores software para determinar qué sintetizador reunía una serie de características deseadas. Entre los sintetizadores estudiados destacan *Renoise, Fluidsynth, Timidity, Ardour* y *Pure Data*. Las referencias bibliográficas de estos se corresponden con el [14], [8], [16], [3] y [28].

Una última parte de investigación, pero no por ello menos importante, ha sido la búsqueda de tutoriales, libros y documentos sobre programación C y C++, tanto en la biblioteca digital de la universidad de Jaén, como en numerosas direcciones web, para poder alcanzar los objetivos del trabajo. Esta información se corresponde con las referencias bibliográficas [1], [2], [6], [9], [20], [21], [23], [24] y [25].

2.2.2. Herramientas utilizadas en la creación del trabajo

La realización del trabajo se ha llevado a cabo en un ordenador portátil con dos sistemas operativos diferentes. El presente documento se ha realizado en Windows 7, mientras que el programa ha sido desarrollado en Linux 14.04 LTS.

La memoria ha sido desarrollada mediante el procesador de textos Microsoft Office Word junto con un software llamado *Dia* para la realización de los diagramas de flujo.

El código desarrollado se ha realizado con Code::Blocks, un entorno de desarrollo integrado libre y multiplataforma para el desarrollo de programas en lenguaje C y C++.

Para las pruebas realizadas durante el desarrollo del trabajo con distintos archivos de audio, se ha hecho uso de un conversor de audio llamado *conversor de audio*, ubicado en el Centro de software de Ubuntu. También se ha utilizado un software libre llamado *Audacity* para colocar distintas pistas en cada canal (izquierdo y derecho) y así poder observar retardos o cambios de ritmo.

2.2.3. Creación del programa

Para la creación del programa se ha seguido una serie de pasos que ha permitido el cumplimiento de los objetivos marcados.

- Creación de un modelo cliente/servidor cuya función es enviar eventos (notas MIDI) a un sintetizador (servidor), ya sea software, como el fluidsynth, usado durante todo el desarrollo del código para ir haciendo pruebas o a una tarjeta de sonido.
- Creación de un hilo encargado de dar formato a los mensajes MIDI obtenidos del archivo MIDI, para que puedan ser leídos por el servidor, el cual se correspondería con el sintetizador. Este hilo también se encarga de detener la reproducción de un MIDI o de continuar con la reproducción en función del mensaje que sea recibido.
- Creación de un hilo encargado de despertar el sistema cuando reciba un determinado evento, ya sea de Stop, Play o Tempo.
- Sincronización de las distintas hebras que forman el programa y depuración.
- Realización de pruebas con archivos MIDI de distinta duración y con variedad instrumental para solucionar posibles errores.

3. OBJETIVOS

El objetivo fundamental de este trabajo fin de máster es familiarizarse con el estándar MIDI, para crear un instrumento virtual que permita hacer uso del envío de eventos que especifiquen la notación musical, tempo, stop o play, a un sintetizador, además de alcanzar un determinado nivel de conocimiento en el lenguaje de programación C y C++, para así poder desarrollar todo el código del programa. Para lograr lo citado se tiene que:

- Conocer los fundamentos teóricos del trabajo.
- Conocer para qué tipo de dispositivos se implementa dicho programa.
- Configuración de un entorno de desarrollo para poder implementar el código del programa.
- Uso de una biblioteca MIDI que sea capaz de obtener toda la información de un determinado archivo MIDI.
- Enviar los eventos MIDI, extraídos del archivo MIDI con su correspondiente biblioteca, a un sintetizador software o físico haciendo uso de un modelo TCP/IP.
- Sintetizar cualquier tipo de instrumento que permita el estándar MIDI.
- Usar interrupciones del sistema para que cuando se esté reproduciendo un audio MIDI, el sistema despierte si recibe un determinado evento, ya sea una nueva nota o cambio de tempo.
- Cambiar el tempo de la melodía en tiempo real.
- Detener la reproducción en cualquier instante a través del envío de un mensaje específico.

- Continuar la reproducción cuando se requiera a través de un mensaje específico.
- Pasar la duración en tiempo real de un archivo MIDI a tiempo de beat, para que cuando se detenga la reproducción, esta pueda ser continuada en un instante de beat deseado.
- Hacer uso de mensajes OSC para el envío de cualquier evento.

Cabe destacar otros objetivos secundarios muy importantes para el correcto desarrollo y funcionamiento del programa, así como para la realización de distintas pruebas con el fin de detectar posibles errores. Estos son:

- Comprender el seguimiento de partituras en tiempo real que sea capaz de tomar decisiones hacia el futuro del tempo de la interpretación.
- Comprender el entrenamiento del sistema para determinar a priori las propiedades espectrales de cada nota/acorde de la partitura.

4. MATERIALES Y MÉTODOS

En este apartado se hace un breve repaso al origen, historia y como ha ido evolucionando el sintetizador con el paso de los años. También se hace referencia al sintetizador software utilizado a lo largo de todo el trabajo, el cual ha sido Fluidsynth. Se ha elegido este sintetizador y no Timidity porque presenta una versión más reciente, es decir, está más actualizado.

Por último, se explica con mayor detalle la biblioteca MIDI que se ha utilizado, el cambio de formato de los mensajes, las interrupciones del sistema implementadas, control de tempo en tiempo real, mensajes de stop y play, y la sincronización de todas las hebras que han sido utilizadas, junto con sus correspondientes diagramas de flujo.

4.1 ESTADO DEL ARTE

4.1.1. Sintetizador

Un sintetizador de sonidos es un instrumento musical electrónico que genera señales eléctricas convertidas a sonidos a través de bocinas o auriculares. Los sintetizadores pueden imitar otros instrumentos o generar nuevos timbres. Usualmente son ejecutados a través de un teclado, pero pueden ser controlados con varios tipos de dispositivos de entrada, incluyendo secuenciadores, controladores, fingerboards, sintetizadores de guitarra, instrumentos electrónicos de viento, y baterías electrónicas. Los sintetizadores que no cuentan con algún tipo de controlador son llamados *módulos*, y son controlados a través de MIDI o control de voltaje.

Los sintetizadores usan varios métodos para generar una señal. Entre las más populares técnicas de síntesis están: síntesis aditiva, síntesis substractiva, de modulación de frecuencia, de modelado físico, modulación de fase y síntesis basada en samples.

4.1.2. Historia y evolución

Unos de los primeros instrumentos eléctricos, el "telégrafo musical", fue inventado por el ingeniero eléctrico Elisha Gray quien solicitó una patente el 27 de enero de 1876. Accidentalmente descubrió la generación de sonidos a través de la propia vibración de un circuito electromagnético, e inventó un oscilador básico de una nota. Este "telégrafo musical" usó una lengüeta de acero con oscilaciones creadas por electro magnetos transmitidas por una línea telegráfica. Gray también implementó una bocina sencilla en los siguientes modelos. Constaba de un diafragma que vibraba en un campo magnético, haciendo al oscilador audible.

Figura 2. Telégrafo musical

En 1887, Thaddeus Cahill inventó el telarmonio (o dinamófono) el cual usaba dinamos. Era capaz de realizar síntesis aditiva como el órgano Hammond, el cual fue inventado en 1934. Sin embargo, el negocio de Cahill no fue exitoso debido a varias razones como el tamaño del sistema, la rápida evolución de los electrónicos, diafonía en la línea telefónica etc.

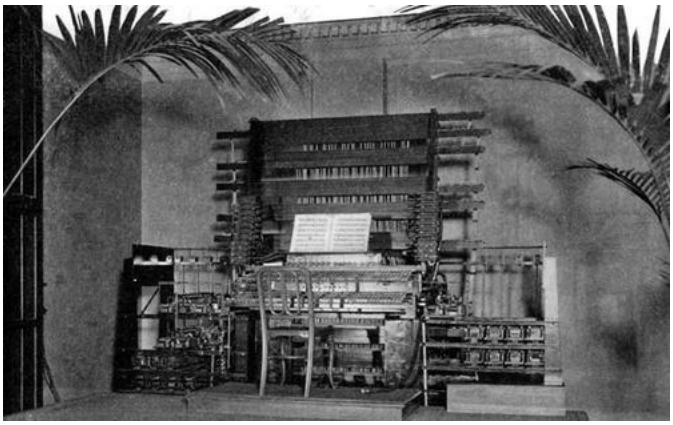

Figura 3. Telarmoni

En 1906, el ingeniero estadounidense Lee De Forest, inventó el primer bulbo amplificador, llamado Audión. Esto llevó al desarrollo de nuevas tecnologías del entretenimiento, incluyendo la radio y el cine sonoro.

En la década de 1920, Arseny Avraamov desarrollo varios sistemas de sonido gráfico. En 1938, el ingeniero de la URSS Yevgeny Murzin diseñó uno de los primeros sintetizadores aditivos en tiempo real usando optoelectrónicos.

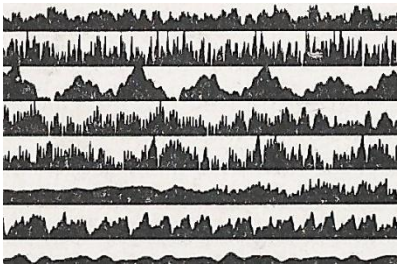

Figura 4. Sonido gráfico

Los primeros sintetizadores polifónicos fueron desarrollados en Alemania y en Estados Unidos. El órgano *Warbo Formant* fue desarrollado por Harald Bode en Alemania en 1937, era un teclado de cuatro teclas/voces con dos filtros de formantes y un control dinámico de envolventes. El "Novachord" de la marca Hammond fue lanzado en 1939, era un teclado electrónico que usaba doce sets de osciladores por octava con divisores de octavas para generar el sonido, un resonador con banco de filtros y un controlador de envolventes dinámico.

Figura 5. Teclado Novachord

Moog es conocido como el primer diseñador de sintetizadores que popularizó la técnica de "control de voltaje" en instrumentos musicales analógicos, el cual construyó su primer prototipo entre 1963 y 1964.

Figura 6. Sintetizador moog

En 1978, el éxito del Prophet-5, un sintetizador polifónico con un microprocesador controlado por teclado, ayudó a la adaptación de los sintetizadores hacia un entorno más moderno, en comparación a las unidades modulares y más enfocado a instrumentos con teclado pequeños. Este factor ayudó a acelerar la integración de los sintetizadores en la música popular.

Figura 7. Sintetizador prophet-5

Lo sintetizadores modernos usualmente tienen una apariencia de pianos pequeños, algunos con perillas y botones adicionales. Estos se volvieron fáciles de integrar y sincronizar con otros instrumentos electrónicos y controlados con la introducción del Musical Instrument Digital Interface (MIDI) en 1983.

El estándar software General MIDI (GM) fue concebido en 1991 para servir como un camino consistente para describir un set de más de 200 sonidos (incluyendo percusiones) disponibles en las PC para playback o partituras musicales.

4.1.3. Fluidsynth

A continuación se explica en detalle el sintetizador software utilizado durante la ejecución del trabajo junto con sus características y posibles mensajes.

FluidSynth, es un Sintetizador software de código abierto que convierte notas MIDI en una señal de audio haciendo uso de la tecnología SoundFont, sin la necesidad de una tarjeta de sonido compatible con SoundFont. FluidSynth puede actuar como un dispositivo virtual MIDI, capaz de recibir datos MIDI desde cualquier programa y transformar estos en audio. También puede leer ficheros MIDI estándar (SMF) directamente. Puede convertir un fichero SMF directamente en un fichero de audio más rápido que en tiempo real. Todas estas características permiten a Fluidsynth ser usado en los siguientes casos:

- Sintetizar datos MIDI desde otra aplicación directamente a los altavoces.
- Sintetizar datos MIDI desde otra aplicación, registrando la salida a un archivo de audio.
- Reproducir un archivo MIDI a través de los altavoces.
- Conversión de un archivo MIDI a un archivo de audio digital.

El tamaño cargado por los bancos SoundFont está limitado por la cantidad de memoria ram disponible. Existe una interfaz gráfica de usuario para FluidSynth llamada Qsynth, la cual es también de código abierto.

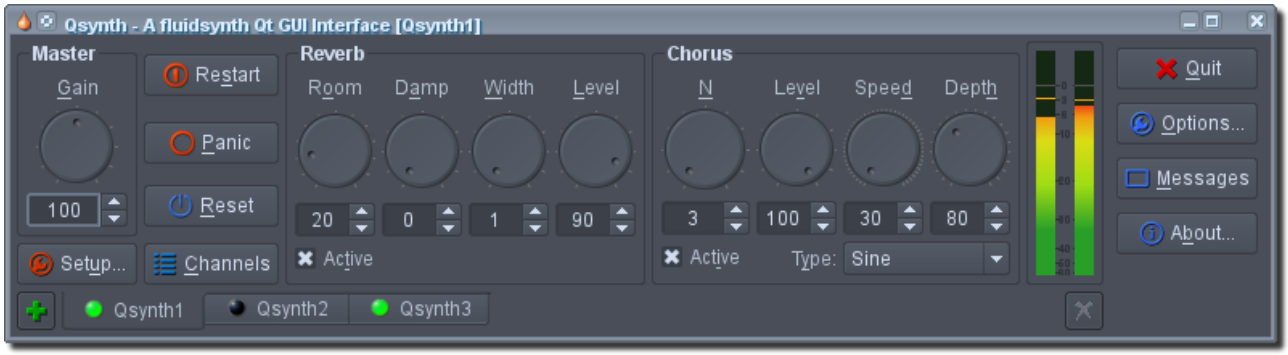

Figura 8. Qsynth (GUI Fluidsynth)

A continuación se muestra una lista con todas las opciones, algunos grupos de ajustes y mensajes que permite Fluidsynth a través de la línea de comandos. No todos ellos han sido utilizados en el desarrollo del trabajo.

➢ **OPCIONES**

- -a, --audio-driver=[label]

  Indica el driver del audio a usar.

- -C, --chorus

  Activa o desactiva el coro.

- -c, --audio-bufcount=[count]
  Número de buffers de audio
- -d, --dump
  Volcado de eventos MIDI entrantes y salientes a la salida estándar.
- -f, --load-config
  Configuración de la línea de comandos.
- -F, --fast-render=[file]
  Almacena los datos de audio de un archivo MIDI sin procesar en un archivo.
- -G, --audio-groups
  Define el número de nodos de audio LADSPA.
- -g, --gain
  Establece la ganancia (0 < ganancia < 10).
- -h, --help
  Muestra un resumen de la ayuda.
- -i, --no-shell
  No lee comandos de la terminal.
- -j, --connect-jack-outputs
  Intenta conectar los jacks de salida a los puertos de salida.
- -K, --midi-channels=[num]
  Numero de canales MIDI. Por defecto 16.
- -L, --audio-channels=[num]
  Número de canales de audio estéreo. Por defecto 1.
- -l, --disable-lash
  No conecta al servidor LASH
- -m, --midi-driver=[label]
  Nombre del driver MIDI a usar.
- -n, --no-midi-in
  No crea un driver MIDI para leer eventos de entrada.
- -O
  Formato de archivo de audio para la representación rápida
- -o
  Define un tipo de ajuste. Ver ajustes Fluidsynth
- -p, --portname=[label]
  Establece el nombre del Puerto MIDI
- -R, --reverb
  Reverberación on o off.
- -r, --sample-rate

Establece la frecuencia de muestreo.

- -s, --server
  Arranca fluidsynth como un proceso servidor.
- -V, --version
  Muestra la version del programa
- -T, --audio-file-type
  Tipo de archivo de audio para la representación rápida.
- -v, --verbose
  Pinta los mensajes de eventos MIDI.
- -V, --version
  Muestra la versión del programa.
- -z, --audio-bufsize=[size]
  Tamaño de cada buffer de audio.

➢ **COMANDOS DE CONSOLA**

General

- help
  Pinta un resumen de los comandos principales.
- help help
  Pinta una lista de otros temas de ayuda.
- quit
  Salir del sintetizador.

Soundfonts

- load *nombre_archivo*
  Carga un soundfonts.
- Unload *número*
  Descarga un soundfont. El número es el índice del soundfont en la pila.
- fonts
  Lista de actuals soundfont en la pila.
- inst *número*
  Muestra los instrumentos disponibles para el soundfont.

Mensajes MIDI

Estos mensajes han sido los más utilizados en la implementación del código, ya que son los que permiten que el sintetizador reproduzca cualquier tipo de instrumento MIDI con una determinada nota y en el canal requerido. Para tener una mayor comprensión de ellos, abajo se muestra las características y campos de cada uno de ellos.

- Noteon *canal nota intensidad*
  Mensaje encargado de enviar al sintetizador la activación de una nota indicando el canal e intensidad de esta.
- noteoff *canal nota*
  Mensaje encargado de enviar al sintetizador el apagado de una determinada nota, indicando el canal correspondiente.
- cc *canal control valor*
  Como se ha comentado anteriormente en el apartado de la especificación MIDI, más en concreto en el apartado *2.1.6.7. Los mensajes de control change (cambio de control)*, estos mensajes permiten aplicar diferentes tipos de controles al canal deseado con el valor requerido.
- prog *canal número*
  Este tipo de mensaje también aparece detallado en el apartado *2.1.6.7. Los mensajes de control change (cambio de control)*, mediante el cual, se puede elegir el instrumento que se requiera para el canal deseado.
- select *canal soundfont banco prog*
  Mensaje que permite hacer una combinación de un banco de sonido y un cambio de programa.
- channels
  Muestra todos los canales en los que aparece un instrumento.

Síntesis de audio

- gain *value*
  Establece la ganancia maestra (0 < ganancia < 5)
- interp *num*
  Elige el método de interpolación para todos los canales.
- interpc *chan num*
  Elige el método de interpolación para un canal.

Reverberación

- reverb
  Reverberación on u off.
- rev_setroomsize *num*
  Cambio del tamaño de la reverberación de la sala.
- rev_setwidth *num*
  Cambia la anchura de la reverberación.
- rev_setlevel *num*
  Cambia el nivel de reverberación.

Aquí quedan mostrados los comandos más usuales con los que trabaja Fluidsynth para poder personalidad la reproducción de una interpretación MIDI.

4.1.4. Mensajes OSC

OSC es el acrónimo de Open Sound Code. Se trata de un protocolo para la comunicación entre ordenadores, sintetizadores musicales y otros dispositivos multimedia inspirado en la tecnología de las redes. El protocolo tiene algunas ventajas como por ejemplo la independencia del medio de transmisión y la flexibilidad para transportar cualquier tipo de datos. Pensado originalmente para la comunicación de instrumentos musicales (como en el caso del protocolo MIDI ya explicado).

El medio de transporte por excelencia son las redes de ordenadores y para obtener velocidad y simplificar las comunicaciones, generalmente se usa el protocolo de transporte UDP (User Datagram Protocol) aunque se pueden usar otros protocolos como por ejemplo el TCP (Transmission Control Protocol).

Características

Las características principales de los mensajes OSC son:

- Ampliable, dinámico. Esquema de nombres simbólicos tipo URL
- Datos numéricos simbólicos y de alta resolución.
- Lenguaje de coincidencia de patrones (pattern matching) para especificar múltiples receptores de un único mensaje
- Marcas de tiempo (time tags) de alta resolución.
- Mensajes “empaquetados” para aquellos eventos que deben ocurrir simultáneamente.
- Sistema de interrogación para encontrar dinámicamente las capacidades de un servidor OSC y obtener documentación.

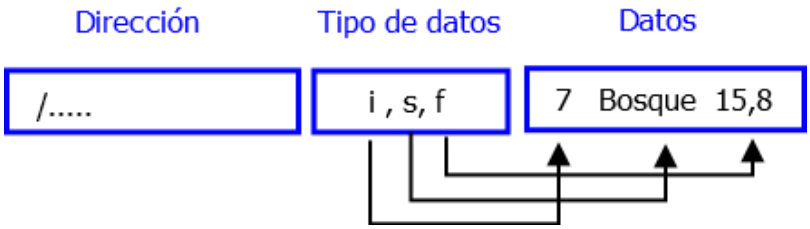

Figura 9. Estructura mensaje OSC

Tipos de mensajes

La base de la comunicación OSC se encuentra en los mensajes. Estos pueden ser de dos tipos:

- Mensaje único.
- Paquete de mensajes. Se trata de un contenedor que puede alojar varios mensajes únicos.

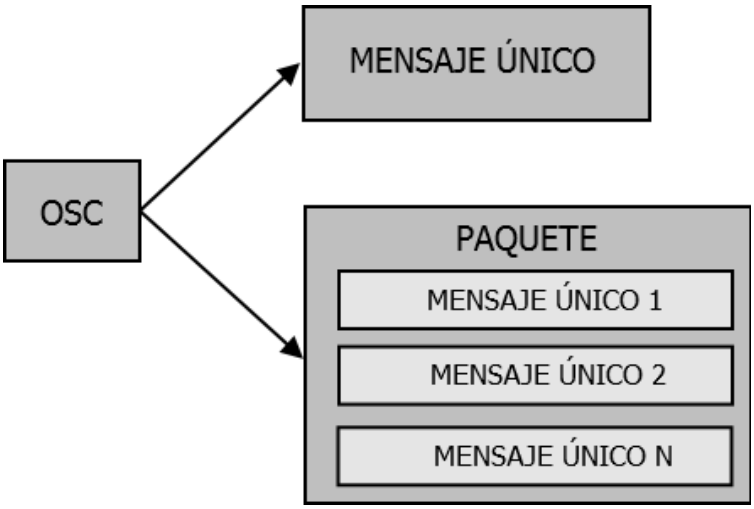

Figura 10. Mensajes OSC

Dirección

La primera parte de un mensaje OSC es la dirección y consiste en una cadena que inicia con el símbolo “ / ”. Presenta un gran parecido a un sistema tipo URL de internet, permitiendo la navegación en una estructura jerárquica. Generalmente, la dirección se usa para identificar el mensaje.

Cadena de identificación de los datos transportados

La segunda parte de un mensaje OSC es una cadena que comienza con el símbolo ” , ” y está compuesta por letras que sirven para identificar los tipos de datos que el mensaje transporta. Cada letra representa un tipo de dato (cadenas, números enteros, floar…etc).

TIPO	DESCRIPCIÓN	ESPACIO OCUPADO
i	Número entero con signo	4 bytes
f	Número con coma flotante	4 bytes
s	Cadena de caracteres	Variable
b	Matriz de bytes	Variable
T	Verdadero sin datos	Ninguno
F	Falso sin datos	Ninguno
N	Nada sin datos (NULL)	Ninguno
I	Impulso sin datos (bang / trigger)	Ninguno
t	Código de tiempo en formato NTP	8 bytes

Tabla 5. Tipos de datos estándar

En la tabla 5 se puede observar los tipos de datos con los que trabaja el protocolo de comunicaciones OSC.

Datos

La tercera parte del mensaje está formada por los datos, uno después de otro y sin ningún byte de separación entre ellos. Los números generalmente ocupan cuatro bytes, mientras que las cadenas de caracteres y las matrices de bytes pueden tener un largo variable pero siempre un valor múltiplo de cuatro.

Cadenas de caracteres OSC

Son secuencias de caracteres ASCII. Se identifican con la letra " s " minúscula del campo Type Tag. El dato tiene que respetar dos condiciones: la primera es que en el final de la cadena debe de haber por lo menos un byte con valor 0 (/0) indicando el final de la misma, y la segunda es que el largo de la cadena completa en byte debe de ser un múltiplo de 4, por ejemplo, 4, 8, 12, 14…etc bytes.

A continuación se muestra como se formaría una cadena de caracteres OSC:

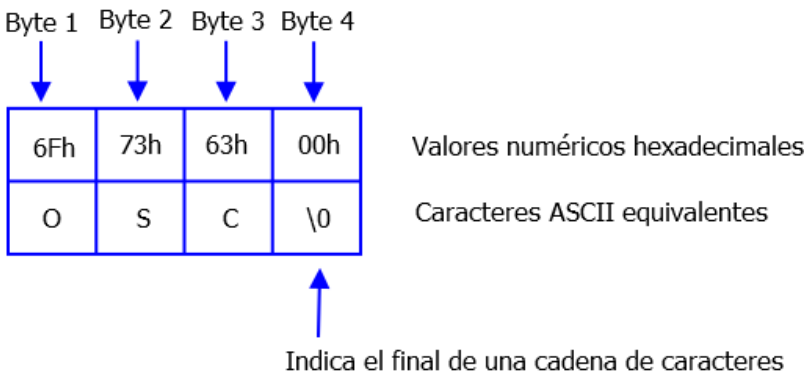

Figura 11. Cadena OSC

Blob (matriz de bytes)

El tipo de dato Blob permite transportar una matriz de bytes de cualquier dimensión. Está constituido por dos partes. La primera parte es un número entero que ocupa 4 bytes (int32) y que indica la dimensión de la matriz. La segunda parte es la matriz de byte propiamente dicha. Si la cantidad de elementos de la matriz no es múltiplo de 4 se deben agregar al final de la matriz unos bytes con contenido 0 para alcanzar este valor múltiplo. Los byte agregados no afectan al número de la dimensión de la matriz. El tipo blob es útil cuando se deben transmitir un grupo de bytes que pueden contener como dato 0 como por ejemplo un diseño bitmap y que no sería posible hacerlo con una dato del tipo string.

En la figura 12 se observa un ejemplo de dato Blob constituido por una matriz de 6 bytes. Los primeros 4 bytes representan un número entero (Int32) que indica la cantidad de byte que componen la matriz, en este caso 6. Los siguientes bytes representan la matriz. El tercer byte de la matriz tiene como contenido 0, si hubiéramos usado un dato del tipo cadena (string) para transportar esta matriz, este byte habría creado un conflicto porque sería interpretado como el final de la cadena. Los dos últimos byte con valor 0 sirven para “redondear” el largo del dato a un múltiplo de 4.

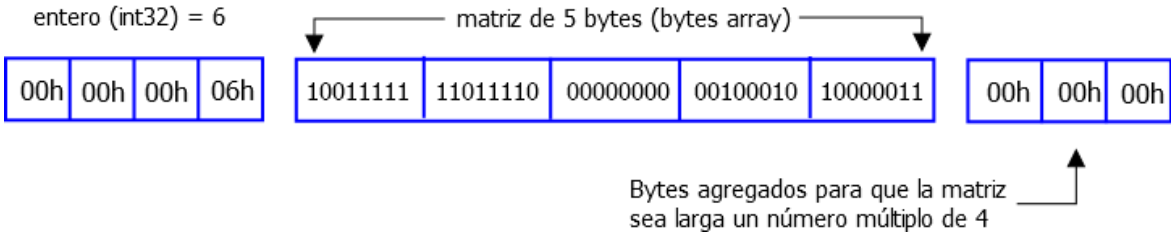

Figura 12. Matriz de bytes

4.2 PROGRAMA DESARROLLADO

El programa llevado a cabo en este trabajo fin de máster ha sido la realización de un instrumento virtual en lenguaje de programación C y C++, que trabajando junto con la especificación MIDI, permite reproducir cualquier nota en escala MIDI representada por el instrumento deseado. También permite la opción de realizar un cambio de tempo en tiempo real a través de mensajes OSC, así como el envío de los eventos stop y play para detener la reproducción del archivo MIDI o continuarla en el instante de beat deseado.

Las tareas que se han realizado basándose en los objetivos iniciales marcados son:

- Manejo y uso de una biblioteca MIDI para obtener toda la información de cualquier archivo con este formato.
- Reproducir notas MIDI en el sintetizador software utilizado (Fluidsynth), mediante el envío de eventos de activación de nota a través de un modelo cliente/servidor.
- Sintetizar cualquier nota en escala MIDI con el instrumento requerido y en el canal

correspondiente.

- Ejecutar cualquier cambio de programa que aparezca en la información adicional de un archivo MIDI, ya sea coros, cambio de volumen, reverberación, modulación...etc.
- Cambio de tempo en tiempo real cuando se reciba un evento que así lo indique.
- Detener la reproducción del audio cuando se reciba un mensaje específico con esta información. Evento stop.
- Cálculo de pasar la duración real de un archivo de audio a tiempo de beat.
- Continuar la reproducción del MIDI por donde se quedó cuando se recibió un evento stop o continuarla por el tiempo de beat deseado. Este tipo de mensaje se corresponde con el evento play.
- Implementación de un modelo cliente/servidor para el envío y recepción de eventos de tempo, play y stop durante las pruebas.
- Lectura de un archivo .txt (pruebas) donde aparece la información ordenada en tres columnas:

    1ª Instantes de tiempo en el que se debe enviar un determinado evento.

    2ª Cambios de tempo en cada instante, evento stop o evento play.

    3ª Beat al que se desea acceder cuando se recibe un evento play.

Una vez terminadas las pruebas, todos los eventos de tempo, play y stop son enviados mediante mensajes OSC, dejando a un lado la lectura del archivo txt donde aparecía dicha información.

A continuación se entra más en detalle punto por punto, del trabajo realizado.

4.2.1. Biblioteca MidiFile

La biblioteca MIDIFILE es el punto de partida de este trabajo, ya que de alguna manera, se necesita sacar u obtener toda la información que contiene un archivo MIDI. Se trata de una biblioteca para analizar archivos MIDI estándar, con una gran variedad de características que permite manipular todo el contenido MIDI de una forma u otra.

Esta biblioteca está formada por seis clases, que a su vez, estas presentan una serie de métodos para realizar diferentes operaciones.

- **MidiFile**

  Interfaz principal para la lectura/escritura de archivos MIDI estándar. La clase MidiFile aparece como un array dimensional, la primera dimensión lista los tracks mientras que en la segunda dimensión aparecen listados los eventos MIDI.
- **MidiEventList**

  Es una estructura de datos que gestiona la lista de eventos MIDI de una pista.
- **MidiMessage**

  Clase para obtener o establecer determinados parámetros.

- **MidiEvent**

  Hereda de la clase MidiMessage. Presenta opciones temporales.

- **Options**

  Clase auxiliar para las opciones de la línea de comandos.

- **Binasc**

  Clase de ayuda para el análisis de archivos MIDI que permiten la lectura/escritura de archivos MIDI en un formato ASCII, que describe los bytes del archivo MIDI estándar binario.

En este programa, principalmente se ha trabajado con las clases MidiFile y Options. Dentro de la clase MidiFile, se ha hecho uso del método *getTrackCount()*, para obtener el número de pistas, y del método *getTicksPerQuarterNote()*, para obtener los ticks por cuarto de nota y así poder realizar el cálculo correcto para los nuevos tempos recibidos. Dentro de la clase Options se ha hecho uso del método *process()*, encargado de tomar el archivo MIDI pasado a través de consola, y el método *getArgCount()*, cuya función es comprobar la existencia de dicho archivo.

Para poder utilizar la biblioteca, primero debe ser compilada con los siguientes comandos:

- A través de la consola, se navega hasta el directorio de trabajo, el cual será la carpeta contenedora de la biblioteca MidiFile. Una vez en dicho directorio se ejecuta *make library.*
- Esto creará el archivo *'lib/libmidifile.a'* el cual puede ser usado para linkar a programas que usen la biblioteca.

Una vez compilada la biblioteca, se puede hacer uso de todas sus clases y métodos. A continuación se muestra una captura de la línea de comandos con la información que se obtiene al compilar y ejecutar el código encargado de leer y mostrar el contenido de un archivo MIDI.

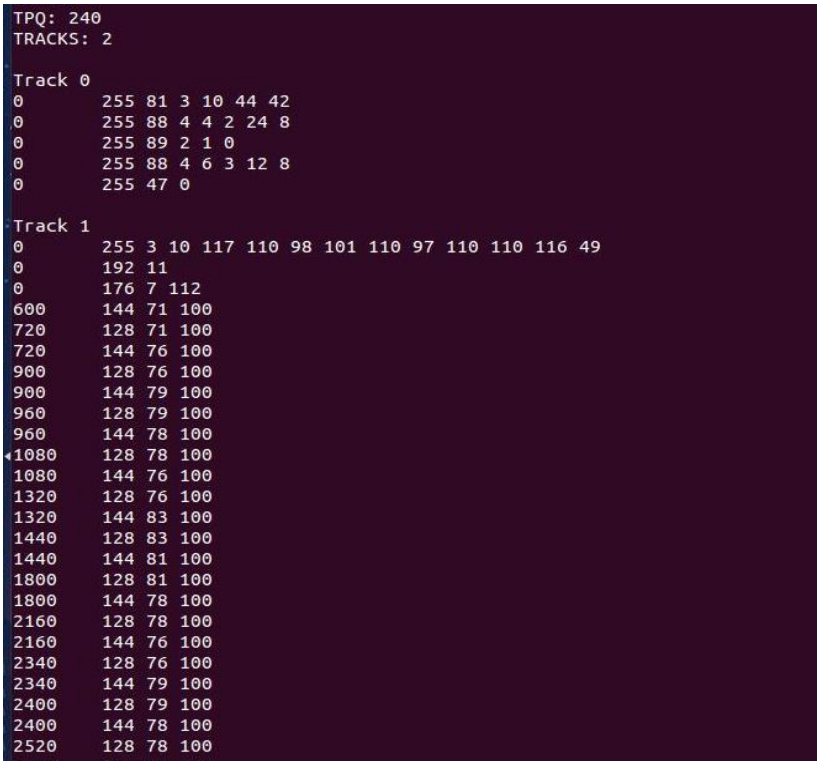

Figura 13. Información obtenida por biblioteca MidiFile

La información que se puede obtener de la figura 13 es la siguiente:

- En primer lugar aparece el TPQ con un valor de 240, es decir, en un cuarto de nota aparecen 240 ticks.
- Después aparece TRACKS con un valor de 2, lo que quiere decir que el archivo de audio presenta 2 pistas.
- Track 0: El track 0 continiene información adicional del MIDI, la cual no repercute en la reproducción de la pista. El primer valor de cada fila (0), indica el instante de tiempo en el que se ejecuta. El siguiente valor a 0 se corresponde con el byte de estado y el resto con los bytes de datos.
- Track 1: En este track, aparece la primera fila con un instante de tiempo 0 e información adicional en el byte de estado y bytes de datos.
- En la segunda fila el instante de tiempo es 0, el byte de estado indica un cambio de programa en el canal 1 y el byte de datos indica que el instrumento que se debe tocar es la caja de música.
- En la tercera fila el instante de tiempo es 0, el byte de estado indica un cambio de programa en el canal 1, el primer byte de datos con valor 7 indica que el control especificado es el volumen y el último byte de datos indica el valor que adquiere el volumen (comprendido entre 0 y 127).
- En la cuarta fila el instante de tiempo ahora es 600, el byte de estado con valor de 144 indica una activación de nota en el canal 1, el primer byte de datos informa sobre la nota en escala MIDI que se ha de tocar, en este caso la 71 y por último, el segundo byte de estado indica la intensidad con la que dicha nota debe de ser tocada.
- En la quinta fila el instante de tiempo es 720, el byte de estado indica un apagado de nota en el canal 1, la cual se corresponde con la nota encendida anteriormente en el instante 600, ya que el valor del primer byte datos indica que esta nota es la 71.

El resto de filas siguen la misma estructura, informando acerca de la activación y desactivación de notas en cada instante de tiempo.

En este caso, para compilar el código a través de consola, se debe indicar la carpeta en la que se encuentran almacenadas las clases y la carpeta donde se localiza la librería creada anteriormente al ejecutar el make library ('*lib/libmidifile.a'*). El comando completo sería:

g++ -o midi main.cpp -I /home/usuario/Escritorio/midifile/craigsapp-midifile-9ee5898/include -L /home/usuario/Escritorio/midifile/craigsapp-midifile-9ee5898/lib -lmidifile

Una vez compilado el código, este se ejecuta a través de la consola también pasándole un audio MIDI. En este ejemplo, el archivo se corresponde con la melodía de la película Harry Potter,

siendo el comando:

./midi /home/usuario/Escritorio/Harry_Potter.mid

Aclarar que esta clase no se encarga de reproducir la información de un archivo MIDI, simplemente la obtiene y muestra con todo detalle. En los siguientes apartados se verá como se trata esta información para que sea reproducida en un sintetizador.

4.2.2. Cambio de formato

Una vez que se consigue el contenido del archivo MIDI, este tiene que ser enviado al sintetizador para que reproduzca las notas que presentan los mensajes MIDI. Pero antes, se tiene que realizar un cambio en el formato de mensaje que se envía al Fluidsynth, para poder ejecutarlos y no pasarlos por alto.

Como se muestra en el apartado *4.1.3. Fluidsynth,* el sintetizador acepta una serie de mensajes, los cuales pueden ser enviados directamente por consola o a través de un cliente TCP/IP, que presentan un formato determinado. Los cuatro mensajes más utilizados, por la frecuencia con la que aparecen en un archivo MIDI son:

- noteon *canal nota intensidad*
- noteoff *canal nota*
- cc *canal control valor*
- prog *canal número*

Como la información conseguida por la biblioteca MIDI presenta los mensajes en decimal, y no indicando en letra el tipo de mensaje que es, se realiza la conversión. A continuación se muestra como aparece la información obtenida por la biblioteca MIDI y como debe de ser presentada a Fluidsynth.

Instante	**Mensaje MIDI**	**Instante**	**Mensaje MIDI**
0	192 11	0	prog 1 11
0	176 7 112	0	cc 1 7 112
600	144 71 100	600	noteon 1 71 100
720	128 71 100	720	noteoff 1 71 100
720	144 76 100	720	noteon 1 76 100
900	128 76 100	900	noteoff 1 76 100
900	144 79 100	900	noteon 1 79 100

Mensajes MidiFile **Mensajes Fluidsynth**

Una vez que el contenido es modificado, se puede enviar al sintetizador en el orden que indique los instantes de tiempo.

4.2.3. Hebras

El programa desarrollado está compuesto por una hebra principal, encargada de recibir los diferentes eventos implementados y de crear dos hebras secundarias, encargadas de la reproducción de los mensajes MIDI y del envío de eventos a la hebra principal para que esta realice las operaciones necesarias. A continuación se detalla el funcionamiento de cada una de ellas.

4.2.3.1. Hebra principal

La hebra principal o main, tiene como objetivo primordial lanzar todo el programa una vez que este es ejecutado. Dentro de esta, se realiza una serie de tareas muy importantes para el correcto funcionamiento del código.

Creación de las hebras secundarias

Cuando el programa es iniciado, la hebra main crea dos hilos secundarios, *hilo_reproducir*, encargado de tratar toda la información del archivo de audio MIDI y su posterior reproducción, y el *hilo_tempo*, que se encargará de enviar los distintos eventos desarrollados (tempo, play y stop), al propio hilo principal, el cual tomará una determinada decisión en función del evento recibido.

Figura 14. Hebras del programa

Servidor TCP/IP

En la hebra principal también aparece alojado un servidor TCP/IP encargado de recibir los eventos, en un determinado puerto, que son enviados desde la hebra secundaria *hilo_tempo.* Una vez que el servidor recibe un mensaje, comprueba su información, y en función de esta, toma una decisión de cambio de tempo, detención de la reproducción del archivo MIDI o continuar con la misma.

A continuación se muestra un diagrama de flujo del proceso llevado a cabo por parte del servidor.

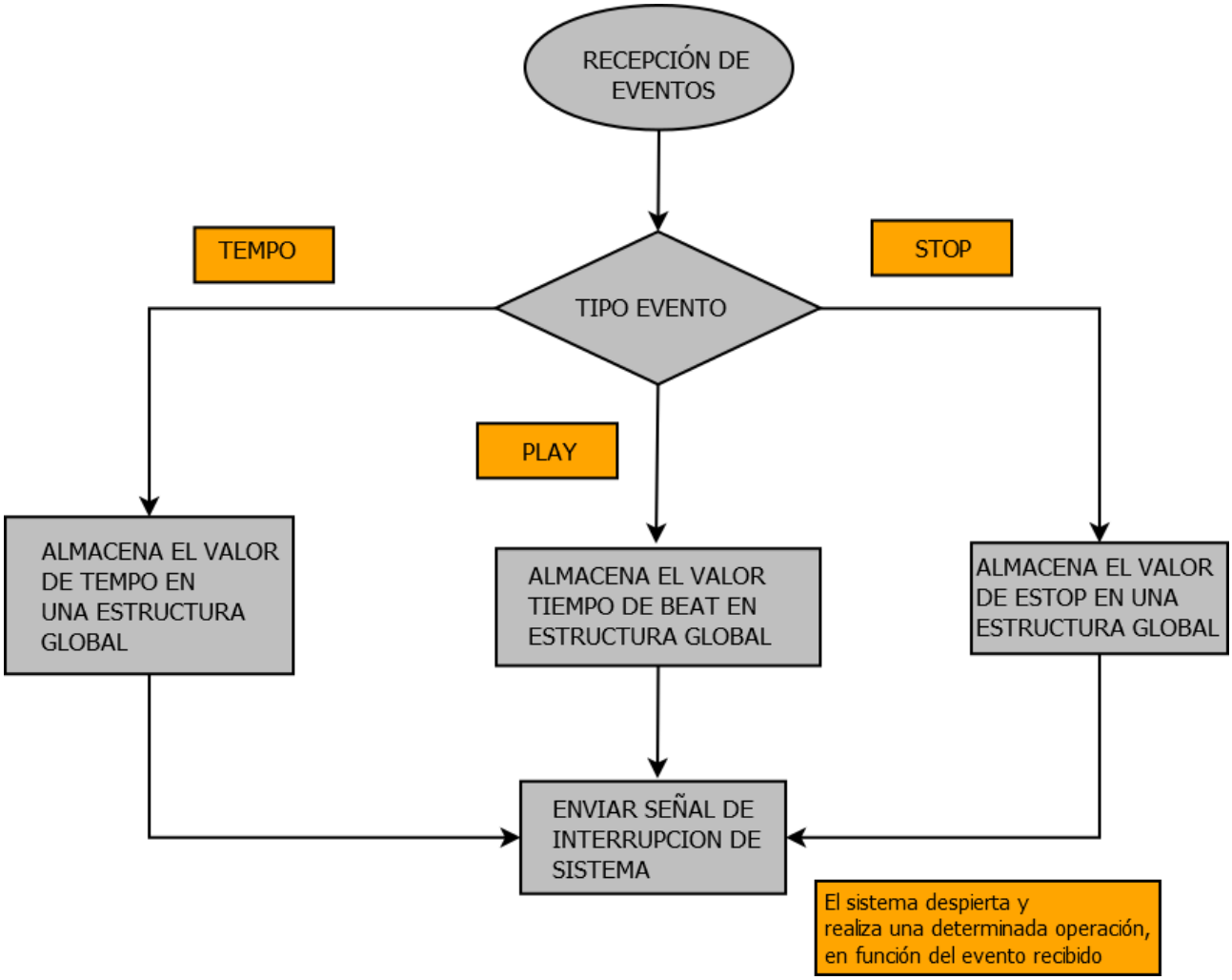

Figura 15. Diagrama de flujo de recepción de eventos

Interrupciones

El hilo principal también es el encargado de enviar señales de interrupción para detener cualquier proceso que esté en ejecución. Estas señales serán lanzadas cuando el servidor TCP alojado en dicho hilo reciba cualquier tipo de evento, ya sea tempo, stop o play.

Para alcanzar este objetivo, se ha utilizado una función propia del kernel de Linux llamada pthread_kill(), la cual permite enviar una determinada señal a la hebra deseada para realizar la acción requerida.

Existen unas treinta y tres señales con objetivos diferentes. En este caso, se ha utilizado la señal SIGTSTP, interrumpiendo la hebra a la que es enviada para poder después continuar desde donde se produjo la interrupción. La indicación del tipo de señal y hebra a interrumpir es la siguiente:

pthread_kill (hilo_a_interrumpir , tipo_de_señal)

El uso de interrupciones es muy importante para evitar acumular grandes retardos, ya que una misma nota musical puede recibir dos o más cambios de tempo antes de que esta finalice, por lo que de alguna forma, se tiene que informar al reproductor de que un nuevo evento ha sido recibido. Si no se hiciese uso de estas señales, los eventos recibidos no serían comprobados ni leídos hasta que la reproducción de la actual nota finalizase.

4.2.3.2. Hebra Reproducir

Esta hebra presenta un papel muy importante por el número de acciones que realiza y por las decisiones que ha de tomar. A continuación se lista las operaciones que lleva a cabo detallando su proceso hasta alcanzar el objetivo final.

Lectura de archivo MIDI

La hebra *hilo_reproducir* comienza haciendo uso de la biblioteca MidiFile, comentada en el apartado *4.2.1. Biblioteca MidiFile*, para leer el contenido del archivo de audio MIDI y posteriormente guardar toda la información. Una vez obtenida la información, se realiza el cambio de formato para que el sintetizador software utilizado reconozca los mensajes MIDI y pueda reproducirlos.

Cliente TCP/IP

Una vez que se tienen los mensajes MIDI con el formato adecuado, se hace uso de un cliente TCP/IP, ubicado en esta hebra, para el envío de todos estos mensajes al sintetizador software o a la tarjeta de sonido correspondiente.

Para que el sintetizador software utilizado reciba los mensajes MIDI, este tiene que ponerse en modo servidor TCP/IP, y así poder recibirlos en un determinado puerto.

Figura 16. Envío de eventos MIDI al sintetizador

Detener reproducción

Cuando un archivo MIDI está reproduciéndose, puede ser detenido en cualquier instante de tiempo a través de mensajes de red con el evento correspondiente o enviándolo directamente a través de consola.

Cálculo de los tiempos de beat

Antes de que comience el envío de eventos al sintetizar por parte del cliente, se realiza el cálculo de los tiempos de beat. Para ello se recorre todo el archivo de audio MIDI midiendo la duración temporal que tiene cada nota, de modo que estas se van agrupando hasta que la duración de todas es igual a la duración temporal que tiene un beat.

La duración temporal de un beat se calcula del siguiente modo:

- Se toma el tempo original del archivo de audio en BPM (beat por minuto), por ejemplo, un tempo muy frecuente es BPM = 120.
- Se conoce cuantos segundos tiene un minuto, 60 seg.
- La duración final de un beat se obtiene haciendo una división muy sencilla:
  - BPM = 120
  - Seg_por_min = 60

$$SegPorBeat = \frac{SegPorMin}{BPM} = \frac{60}{120} = 0.5 \qquad (1)$$

Por lo que en este caso, un beat tendrá una duración de medio segundo.

Continuar reproducción

Al estar la reproducción detenida tras recibir un evento stop, esta puede ser puesta en marcha de nuevo enviando en evento play, continuando por donde se detuvo el audio o en cualquier tiempo de beat deseado.

Cambio de tempo

Para realizar el cálculo de un nuevo tempo, se hace uso de información de configuración que aporta el archivo MIDI. De modo que, cuando se reciba un evento tempo, el valor que presente será multiplicado por el tempo original de la canción (BPM), es decir, si se quiere que la velocidad con la que se reproduce una pieza musical se duplique, el valor que contendrá el evento tempo será igual a 2, en otro caso, si se quiere que la velocidad sea el doble de lenta, el valor de tempo será de 0.5.

Haciendo esta operación, lo que se consigue es que si se multiplica el tempo original del audio por 2, la duración de las notas pasan a tener la mitad de la duración original, en el caso contrario, si se multiplica el tempo por 0.5, la duración de las notas pasará a ser el doble. A continuación se explica la relación que existe entre tempo y duración de las notas con un ejemplo:

- Se toma el tempo del archivo MIDI, por ejemplo BPM = 120
- Se conocen los segundos que tiene un minuto, 60 seg.
- Se calcula la duración temporal de un beat.
- La duración temporal de un beat se pasa a microsegundos, ya que la duración de algunas notas es muy pequeña. De este modo se consigue una gran precisión temporal.
- Con una función de la biblioteca MidiFile, utilizada para leer archivos MIDI, se obtiene los ticks por cuarto de nota, que se corresponden con los ticks que presenta un beat.

$$BPM = 120$$

$$SegPorMin = 60$$

$$SegPorBeat = \frac{SegPorMin}{BPM} = \frac{60}{120} = 0.5$$

$$MicroPorBeat = SegPorBeat\ x\ 1000000 = 500000 \quad (2)$$

$$TicksPorBeat = getTicksPerQuarterNote() = 240 \quad (3)$$

$$MicrosegundosTick = \frac{MicroPorBeat}{TicksPorBeat} = \frac{500000}{240} = 2083.3 \quad (4)$$

En la ecuación (4) se obtiene la duración en microsegundos de un tick, ya que la columna de información temporal de un archivo MIDI aparece en ticks, indicando la duración de reproducción de cada nota. Por lo tanto, un tick tendrá una duración de 2083.3 microsegundos con un tempo de 120 bpm. Si este tempo es multiplicado por 2 para duplicar la velocidad de reproducción del audio, la duración de un tick pasará a ser la mitad, es decir, 1041.6 microsegundos, por lo que las notas verán reducidas su duración de reproducción a la mitad.

Dormir y despertar

Para la reproducción de las notas de un archivo de audio MIDI se hace uso de una función

de C, que suspende la ejecución del hilo en el que se encuentra ubicada. Esta función es *usleep()* y mantendrá la hebra dormida el tiempo que se le indique como parámetro. Este tiempo es el que aparece en la columna de instantes dentro del MIDI, de modo que cuando se toma una determinada nota, también se toma su tiempo de duración. El cliente TCP/IP manda la nota al sintetizador y su duración se le pasa a la función *usleep(),* la cual dormirá el hilo hasta que se consuma el tiempo indicado.

Una vez que la hebra ha sido despertada, se comprueba la siguiente nota, si es un apagar nota, el cliente envía este mensaje al sintetizador para que deje de reproducir la nota anteriormente enviada, sin ningún tiempo para dormir. En el caso de que sea activar otra nota, esta es enviada al sintetizador y su tiempo de duración indicado a la función *usleep()*.

Por lo tanto, la operación que se realiza para reproducir las notas de un archivo MIDI es dormir y despertar continuamente la hebra reproducir para que envíe los mensajes al sintetizador cuando los instantes de tiempo así lo indiquen.

La hebra *hilo_reproducir* también duerme en un segundo caso, y es cuando un evento stop es recibido. Esta se hecha a dormir hasta que un evento play sea recibido de nuevo.

Aclarar que el sistema es despertado en dos ocasiones, la primera es por el agotamiento del tiempo que se le pasa a la función *usleep()* a la hora de reproducir una determinada nota, y la segunda es cuando el servidor de la hebra principal recibe uno de los eventos implementados ya sea tempo, stop o play.

Por último, la figura 17 muestra un diagrama de flujo que se corresponde con el inicio de la hebra reproducir hasta que los mensajes MIDI son enviados al sintetizador:

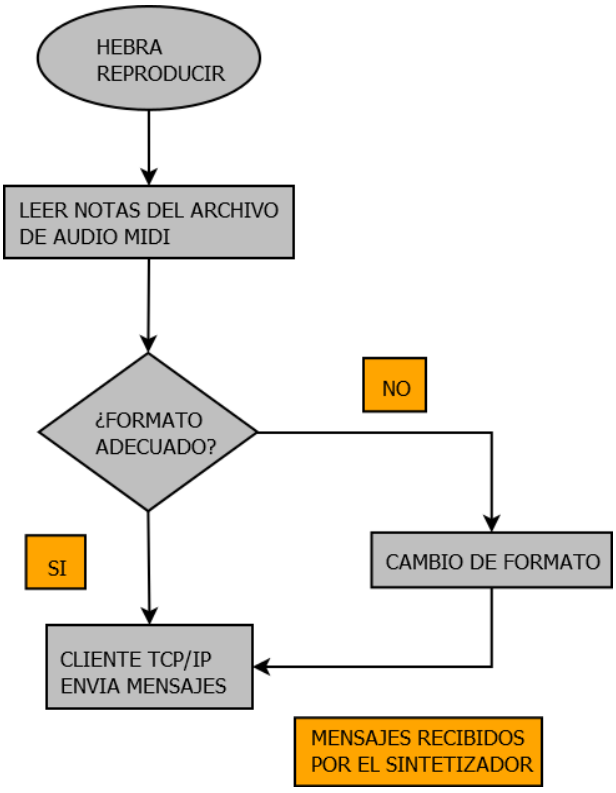

Figura 17. Diagrama de envío de mensajes MIDI

Y el de la figura 18, cuando a esta hebra se le informa de que un evento nuevo ha sido enviado y debe de realizar una determinada operación.

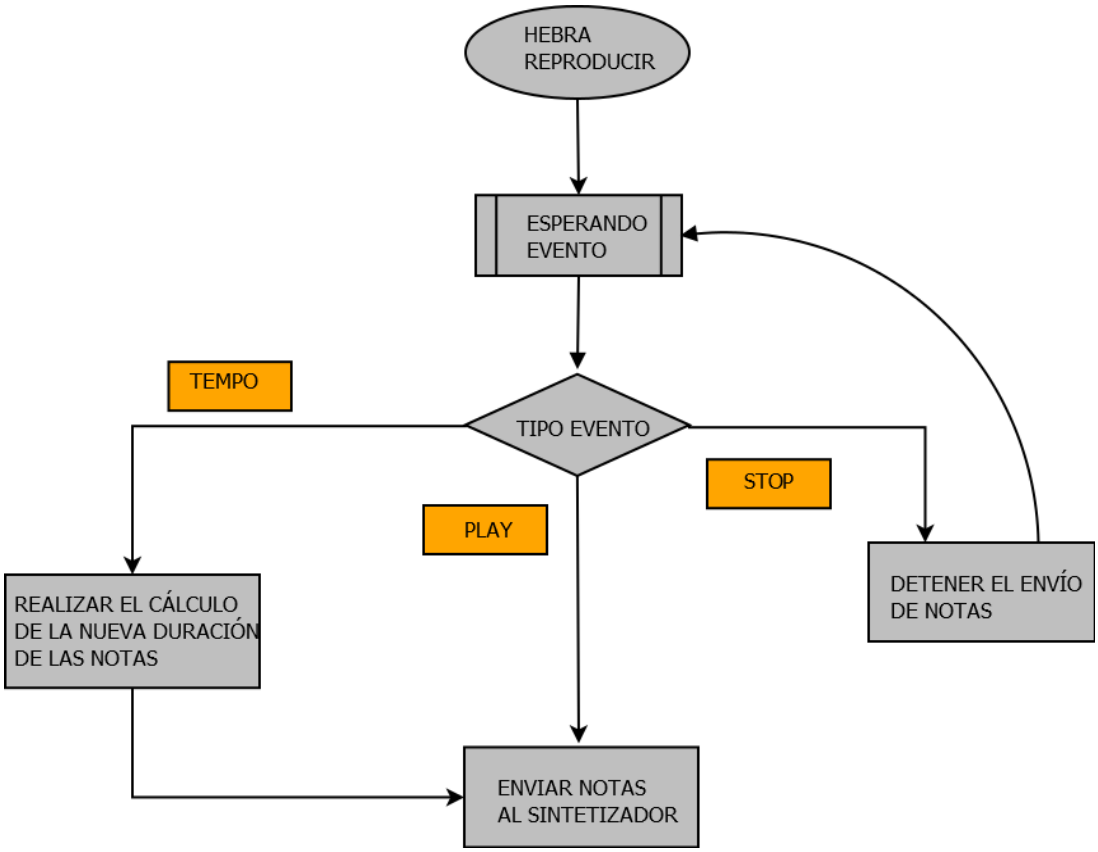

Figura 18. Diagrama de recepción de eventos en el hilo reproducir

4.2.3.3. Hebra tempo

Se trata de un hilo, llamado *hilo_tempo*, encargado de enviar uno de los tres eventos implementados (tempo, play o stop) en un instante de tiempo determinado, al servidor TCP/IP que aparece en el hilo principal. Cuando este recibe uno de los eventos procedentes del hilo tempo, despierta el sistema y realiza el cálculo que sea necesario.

La implementación de esta hebra es muy importante, ya que desde que se comienza a desarrollar el programa, es necesario realizar pruebas con el envío de distintos eventos para comprobar que el código trabajaba correctamente.

4.2.4. Sincronización de hebras

A continuación se muestra una imagen con la sincronización temporal de las hebras utilizadas con una explicación detallada para aclarar su funcionamiento.

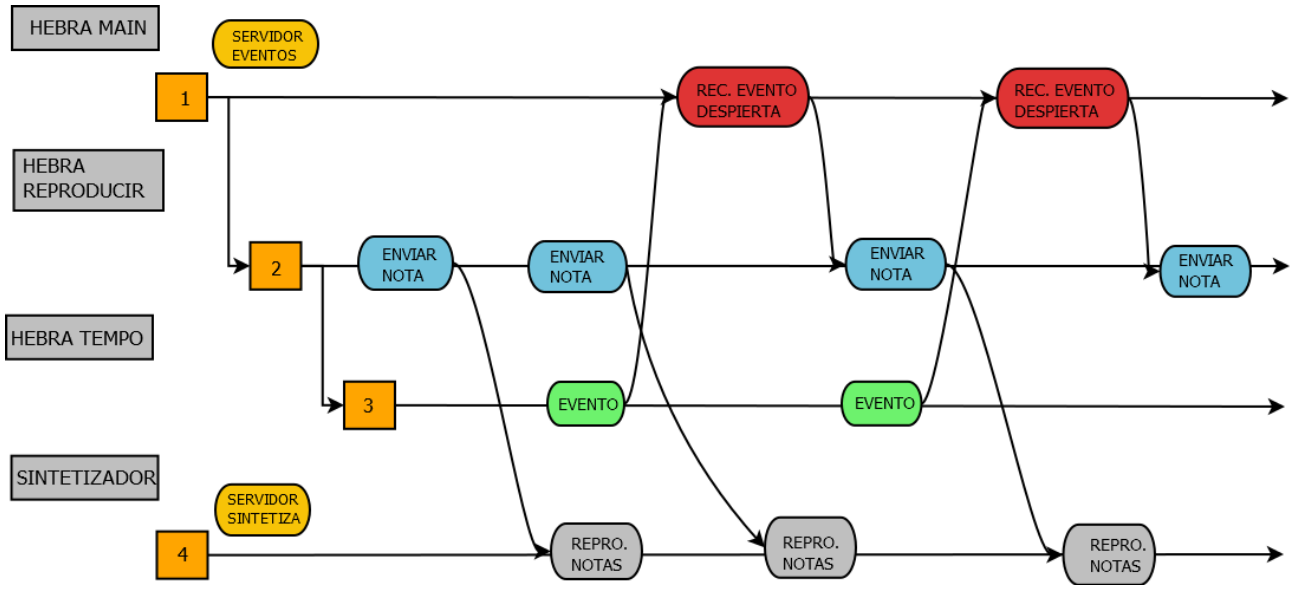

Figura 19. Sincronización temporal de hebras

Como se puede observar en la figura 19, aparecen las tres hebras implementadas junto con el sintetizador encargado de reproducir los mensajes MIDI del archivo de audio.

En primer lugar, se lanza el programa, originando la hebra principal, que a su vez pone en marcha un servidor TCP/IP encargado de recibir los eventos que son enviados desde la hebra *hilo_tempo*. Se lanza también el sintetizador, ya sea una tarjeta de sonido o un sintetizador software. En este último caso, aparecerá como un servidor TCP/IP para recibir los eventos musicales que la hebra *hilo_reproducir* envía.

Una vez que el programa es ejecutado, la hebra principal lanza la primera hebra secundaria, *hilo_reproducir*, la cual se encarga de obtener toda la información del archivo MIDI a través de la biblioteca MidiFile. Cambia el formato de los mensajes para que el sintetizador sea capaz de reconocerlos y mediante un cliente TCP/IP los va enviando en función de los instantes de tiempos indicados. Por ejemplo, el primer mensaje a enviar es noteon 1 74 100, es decir, activar la nota 74 en el canal 1 con una intensidad de 100, y esta se envía al sintetizador. Se obtiene su duración temporal como podría ser 100.000 microsegundos, de modo que la hebra *hilo_reproducir* permanece dormida durante ese periodo de tiempo hasta que es consumido en su totalidad y despierta para comprobar cuál es el siguiente mensaje MIDI. Si el siguiente mensaje es noteoff 1 74 100, cuando sea recibido por el sintetizador apagará la nota 74 anteriormente enviada y pasará a comprobar el siguiente mensaje.

La hebra *hilo_tempo* no se lanza en el mismo instante que la hebra *hilo_reproducir*, ya que la hebra encargada de la reproducción necesita un determinado tiempo para realizar su configuración. Esta consiste en ajustar cada instrumento a su canal, aplicar todos los controles indicados a sus canales correspondientes, como pueden ser coros, reverberación, mayor intensidad de nota… etc, y realizar el cálculo de los tiempos de beat para posteriormente poder indicar en el instante deseado para continuar con la reproducción. Si la hebra *hilo_tempo* se lanzase en el mismo instante, se podría dar el caso de que esta comenzase a enviar eventos sin que la hebra *hilo_reproducir* terminase su configuración, originando errores y que el programa se quede colgado.

Entonces, la hebra *hilo_tempo* envía un evento, por ejemplo tempo=2, para duplicar la velocidad de reproducción de la pieza musical. El servidor alojado en la hebra principal lo recibe e instantáneamente interrumpe el sueño del *hilo_reproducir*, este, al despertar detiene todas las notas que estuviesen reproduciéndose en el sintetizador para comprobar el evento que ha sido recibido por el servidor. Al ver que el evento es un cambio de tempo con valor igual a 2, realiza el cálculo necesario para que la duración del resto de las notas del archivo MIDI pasen a tener la mitad de duración con respecto a la duración original, incluyendo la nota que fue interrumpida cuando el evento fue recibido. Una vez que el cálculo ha sido realizado, se continúa el envío de notas al sintetizador para que siga reproduciéndolas.

Lo mismo ocurrirá si se recibe un evento stop, el hilo principal despierta al *hilo_reproducir* indicándole que detenga el envío de notas al sintetizador, de modo que el *hilo_reproducir* se pone a dormir sin realizar ninguna operación hasta que el *hilo_tempo* envía un evento play, despertándolo de nuevo e indicándole que continúe con la reproducción del audio donde se detuvo o en otro instante de tiempo deseado.

4.2.5. Diagrama con la información del programa

En la figura 20 se puede observar un diagrama en el que se representa todo el flujo de la información del programa.

En primer lugar, cuando se ejecuta el programa, se crea el hilo principal, conocido como main. Como se ha comentado en el apartado anterior *4.2.4. Sincronización de las hebras*, este se encarga de lanzar los hilos secundarios Hilo_reproducir e Hilo_tempos. A su vez, es el responsable de recibir los distintos eventos desarrollados y de despertar el sistema cuando se reciba uno de ellos.

Cuando el hilo principal lanza el Hilo_reproducir, este, haciendo uso de la biblioteca MidiFile, obtiene la información del archivo de audio MIDI, que a través de un cliente TCP/IP envía al sintetizador encargado de reproducir las notas musicales. En este caso se trata de un sintetizador software el cual actuará como un servidor para recibir los eventos que el cliente del Hilo_reproducir envía.

Cuando el Hilo_tempos es lanzado, envía diferentes eventos al hilo principal a través de un segundo cliente TCP/IP, como pueden ser: tempos, play o stop, de modo que cuando el servidor del hilo principal reciba uno de ellos, interrumpirá la ejecución del Hilo_reproducir deteniendo la reproducción del audio MIDI. El Hilo_reproducir comprueba el valor del evento que ha recibido el hilo principal pudiendo realizar tres acciones diferentes: realizando el cálculo de los nuevos tiempos de las notas al recibir un evento tempo, detener la reproducción del MIDI al recibir un evento stop o continuar con la reproducción de la misma si se recibe un evento play.

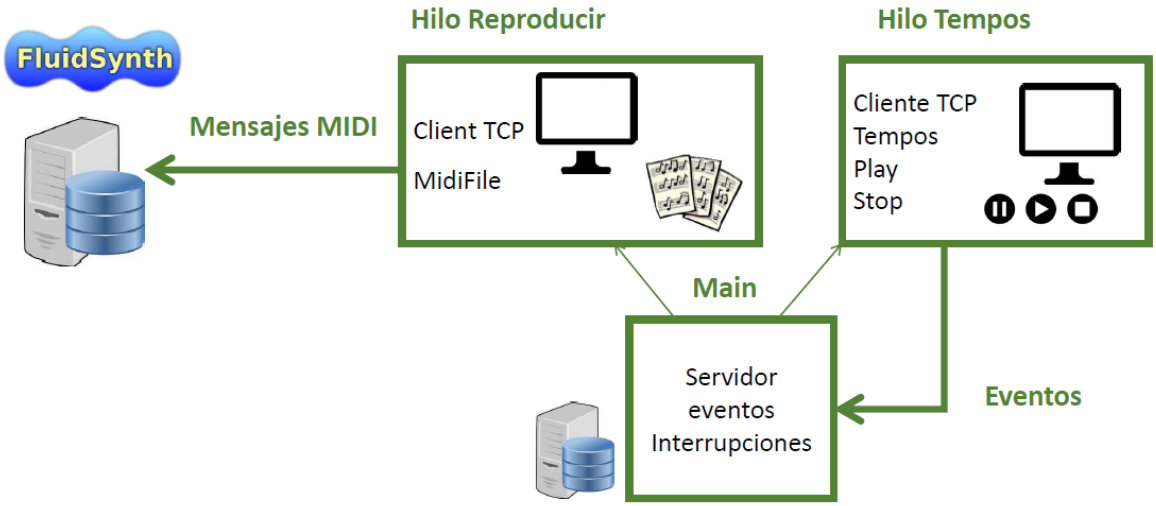

Figura 20. Diagrama programa

4.2.6. Interrupciones y dormir

En los diferentes apartados del punto *4.2 PROGRAMA DESARROLLADO*, se ha visto las funciones que son utilizadas para dormir y despertar el código cuando un nuevo evento es recibido o cuando la duración de la reproducción de una determinada nota finaliza. A continuación se muestra la clase, el método y hebras utilizadas para ver que parte del código se ejecuta cuando el hilo encargado de la reproducción es despertado.

Clase Cliente TCP/IP

- Clase encargada de realizar el envío de notas musicales al sintetizador y de eventos al servidor ubicado en el main.

Método SIG_HANDLER

- Método al que se accede cuando una nota se está reproduciendo y se recibe un determinado evento. El hilo principal (main) despierta al *hilo_reproducir*, saltando el código a este método, encargado de desviar la ejecución de la reproducción del audio a uno de los tres condicionales que presenta el *hilo_reproducir*, calcular nuevo tempo, detener la

reproducción o continuar con esta. Una vez que se ha realizado una de estas tres acciones, se continúa con el envío de notas al sintetizador.

Hebra hilo_tempos

- Hilo encargado del envío de eventos al hilo principal, tempo, stop o play.

Hebra hilo_reproducir

- Hilo encargado de enviar las notas musicales al sintetizador, calcular la duración de las notas tras recibir un nuevo tempo, detener la reproducción del archivo de audio o continuar con la misma.

Main

- Hilo principal encargado de lanzar los hilos secundarios, despertar o interrumpir el *hilo_reproducir* cuando está dormido reproduciendo una determinada nota y de recibir los distintos eventos desde el *hilo_tempos* o a través de los mensajes de red.

Aclarar que el hilo que duerme es el *hilo_reproducir* cuando está reproduciendo una nota. Este despertará cuando la duración de la nota haya finalizado o cuando el hilo principal envíe una interrupción al haber recibido un nuevo evento. La orden de la acción que debe de realizar (cálculo de tempos, stop o play) el *hilo_reproducir* es dada por el método *SIG_HANDLER*.

5. Resultados y discusión

5.1 PRUEBAS

Desde el comienzo del desarrollo del programa, se han ido realizando numerosas pruebas para comprobar que lo implementado funcionaba correctamente.

Una vez que el sintetizador software fue elegido (Fluidsynth), por las características que presenta y por la gran cantidad de mensajes que este puede recibir, se implementó un cliente TCP/IP para ejecutarlo en una consola aparte, encargado de enviar valores de tempos al servidor TCP/IP ubicado en el hilo principal.

Cuando se comprobó que el código respondía correctamente, se pensó en desarrollar otro método de envío de eventos al servidor que fuese más rápido y preciso que estar enviándolos manualmente. Aquí es cuando entra en juego la hebra *hilo_tempo*. Para ello, lo primero que se hace es diseñar en un archivo .txt con la información que se desea enviar junto con los instantes de tiempo para cada uno de estos eventos. Se puede observar a continuación la estructura o el formato que presenta dicho archivo:

ORDEN	INSTANTES TEMPORALES	EVENTOS	BEAT
1	5	0.5	0
2	10	2	0
3	15	1992	0
4	20	1	0
5	25	1992	0
6	30	1993	0
7	35	3	0
8	40	0.5	0
9	45	1	0
10	0	0	0

Tabla 6. Tabla de eventos para pruebas

En la primera columna aparece el orden en el que los eventos van a ser enviados al servidor correspondiente, encargado de procesarlos.

En la segunda columna aparecen los instantes de tiempo, es decir, cada cuanto tiempo se va a realizar el envío de un determinado evento. Los instantes aparecen representados por un número entero o float, correspondiéndose con segundos.

La tercera columna contiene la información del tipo de evento que se va a enviar. Si el mensaje contiene un valor igual a 1992, significa que se trata de un evento tipo stop. Si el valor que presenta esta segunda columna es igual a 1993, se corresponde con un mensaje tipo play. Cualquier otro valor diferente que aparezca en esta columna será tomado como un evento tipo tempo.

En la tercera columna aparece la información de tiempo de beat. Este contenido va a ser indiferente para cuando en la segunda columna haya un mensaje de stop o tempo. En cambio, cuando aparezca un mensaje de play, este valor si será leído, indicando el tiempo de beat en el que se quiere continuar la reproducción.

La última columna informa sobre el tiempo de beat en el que se desea continuar la reproducción tras haber recibido anteriormente un evento de stop. Esta columna solo será leída cuando el evento a enviar sea un play, es decir, cuando tenga un valor igual a 1993. En este caso, el valor asociado es un 0, lo que significa que la reproducción del archivo de audio comenzará desde el principio, en el tiempo de beat igual a 0.

La hebra *hilo_tempo* será la encargada de leer la información de este fichero evento a evento. El funcionamiento es parecido al de la hebra *hilo_reproducir*, de modo que coge el primer

evento, lo envía al hilo principal y se hecha a dormir hasta que el tiempo indicado en la segunda columna sea consumido. Cuando el tiempo se agota, el *hilo_tempo* despierta y pasa a leer el siguiente evento, así hasta que alcanza el mensaje 0 0 0 que indica el final de envío de eventos.

5.2 TRABAJO FINAL OBTENIDO

El trabajo final obtenido ha sido un instrumento virtual desarrollado en C++ capaz de reproducir cualquier archivo de audio MIDI, configurando los diferentes controles que este presenta y pudiendo reproducir cualquier instrumento que permita la especificación MIDI.

Permite recibir tres tipos de eventos diferentes en tiempo real:

- Tempo: Permite variar la velocidad de reproducción de una pieza musical.
- Stop: Permite detener la reproducción del archivo de audio MIDI cuando se desee.
- Play: Permite continuar con la reproducción desde donde esta se detuvo o desde cualquier otro tiempo de beat.

Los distintos eventos con los que permite trabajar el software desarrollado pueden ser recibidos de dos formas diferentes.

- Con el Hilo_tempos: El cual leerá los distintos eventos de un archivo .txt ubicado en la misma carpeta que el programa. La información se almacena por columnas e irá siendo enviada en los instantes de tiempo que se requiera.
- Con mensajes OSC: A través de un código desarrollado en C++ por integrantes del departamento de ingeniería de telecomunicación de la Escuela Politécnica Superior de Linares, basándose en su propio artículo referenciado en la bibliografía número [22]. Este se encarga de realizar un seguimiento de partitura y en base a esto, realizar un envío de tempos a través de red en los instantes de tiempo que se requieran.

5.3 PROBLEMAS ENCONTRADOS

Durante la realización de este trabajo surgieron diferentes problemas, a los cuales se le ha buscado una solución lo más acertada posible.

El primer problema que aparece es al hacer uso de la función *sleep* para suspender la ejecución de los hilos que la contengan. Su funcionamiento a priori parecía correcto, ya que dormía y despertaba el sistema en los instantes de tiempos que se le indicaba. Además, como los primeros archivos MIDI que se utilizaron eran de muy corta duración para comprobar que los reproducía y poco más, no se percibía el aumento de la duración del archivo audio. Sin embargo, cuando se pasó a realizar pruebas con archivos más complejos y pesados, se detectó que la duración aumentaba considerablemente varios segundos, consecuencia del retardo que mete la función *sleep*. Esto se debe a que la función *sleep* necesita un determinado tiempo para despertarse, en la máquina con la que se ha implementado el código era de 120 microsegundos aproximadamente. De modo que

si se reproducía un archivo MIDI con 50.000 eventos, el retardo introducido era de 6 segundos. A esto hay que añadir que todas las máquinas no presentan la misma velocidad de procesado, de modo que este retardo podría ser mucho mayor.

Para solucionar este problema se ha utilizado una función propia de Linux, *gettimeofday()*, la cual se encarga de obtener la hora del sistema. Esta función es lanzada cuando la configuración del sintetizador es finalizada, por lo que antes de enviar el primer mensaje MIDI se toma la hora del sistema. Cuando se realiza todo el cálculo y cambio de formato para enviar la primera nota se vuelve a tomar la hora del sistema y se calcula el tiempo que ha transcurrido desde que se lanzó el primer reloj hasta que se lanza el segundo reloj. La diferencia de tiempos obtenida se le resta a la duración temporal que presenta la nota, de modo que si existiese algún tipo de retraso, este sería eliminado. El proceso se repite para cada una de las notas que presenta el archivo de audio.

Otro problema muy importante surgido es que cuando se enviaba un evento, como puede ser un cambio de tempo, el sistema se esperaba a que la reproducción de la nota que estaba sonando en ese instante finalizase, para comprobar ese nuevo tempo recibido, ya que le hebra encargada de la reproducción estaba dormida. Es decir, si se está reproduciendo una nota que tiene una duración temporal de 1 segundo, y se le envía un evento de tempo con valor igual a 0.01, la duración de esta nota pasaría a ser de 100 segundos, con lo cual, el sistema tendría que esperar esos 100 segundos para comprobar si un nuevo evento ha sido recibido. Esto no es para nada factible ya que es muy normal que en una misma nota se produzcan varios cambios de tempo.

La solución aplicada en este caso ha sido hacer uso de interrupciones del sistema, para que cuando se reciba cualquier tipo de evento, ya sea tempo, stop o play, se interrumpa la reproducción de una nota y se pase a comprobar el evento recibido para actuar en consecuencia del valor indicado, ya sea realizando un nuevo cálculo de tempo, deteniendo la reproducción o continuando con la misma.

Por último, otro problema como ya se ha comentado en el apartado *4.2.2. Cambio de formato*, la información MIDI obtenida del archivo de audio a través de la biblioteca MidiFile, es presentada en binario, de manera que se tiene que crear un nuevo mensaje con esa información para que el sintetizador software utilizado sea capaz de entenderlo y poder ejecutarlo.

6. CONCLUSIONES

Como conclusión principal, se ha obtenido un software capaz de leer la información de un archivo de audio MIDI y enviarla a un sintetizador, de manera que la reproducción de dicha pieza puede ser controlada en tiempo real para adecuarla al músico que la esté utilizando. Esto permite que el ensayo de un músico sea más cómodo y eficiente ya que no tiene que ir fijándose en como otro músico esté ejecutando la pieza musical, simplemente puniendo en marcha el software en una ordenador o tableta podrá ensayar cualquier canción, adecuándose la reproducción del archivo MIDI a la forma en la que el músico interpreta la obra.

7. ANEXOS

7.1 ANEXO I. MANUAL DE USUARIO

El objetivo fundamental del manual de usuario es explicar todos los pasos que han de darse para que el software desarrollado pueda ser ejecutado perfectamente por una persona ajena al proyecto, ya sea para comprobar el funcionamiento de este o por temas de investigación.

En primer lugar, se debe de configurar un entorno de desarrollo para una adecuada compilación del código. En este caso, se ha trabajado con Code::Blocks, un entorno de desarrollo integrado libre y multiplataforma para el desarrollo de programas en lenguaje C y C++. Este programa ha de ser instalado en un sistema operativo Linux, ya que se trata de un software libre, el cual ahorra muchos problemas a la hora de trabajar con librerías y cabeceras que con otro distinto.

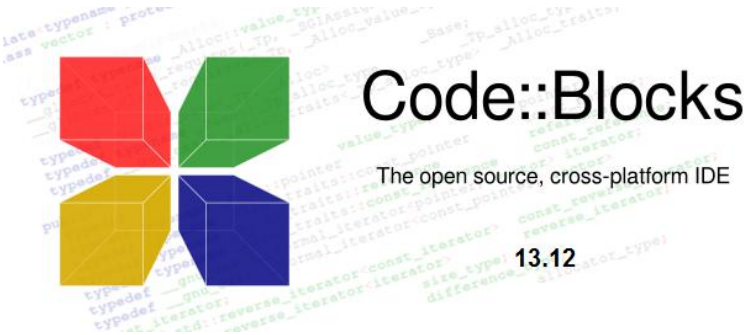

Figura 20. IDE utilizado, Code::Blocks

Una vez que el entorno de desarrollo está preparado, necesitamos una biblioteca MIDI en C que sea capaz de leer los archivos de audio MIDI con los que se quiere trabajar. Existen varias bibliotecas de este tipo, pero por los objetivos marcados en este trabajo se ha utilizado MidiFile, la cual se puede descargar en el siguiente enlace: http://midifile.sapp.org/
Cuando la biblioteca ha sido descargada, se descomprime en una determina carpeta para realizar una serie de operaciones antes de ser utilizada:

- Una vez descomprimida, a través del terminal o consola de comandos, se navega hasta la carpeta contenedora para poder compilarla. Cuando se alcanza dicho directorio se hace uso de una herramienta de gestión de dependencias, en este caso se ha utilizado make, por lo que debe de estar instalado en la máquina con la que se está trabajando. El comando a ejecutar es "*make library".*
- La acción anterior genera un archivo llamado "lib/libmidifile.a", el cual contiene toda la información necesaria para poder procesar archivos de audio MIDI. Esta puede ser lincada a programas que la usen.

En este momento, el código desarrollado en este trabajo fin de máster puede ser compilado fácilmente con el siguiente comando:

g++ -o midi main.cpp -I /home/usuario/Escritorio/midifile/craigsapp-midifile-9ee5898/include -L /home/usuario/Escritorio/midifile/craigsapp-midifile-9ee5898/lib –lmidifile –pthread

- **g++** : Compilador gratuito de C++.
- **o** : Opción que permite crear el ejecutable *midi* del código compilado *main.cpp.*
- **I** : Opción que permite añadir los archivos con los que trabaja la biblioteca Midifile ubicados en este caso en */home/usuario/Escritorio/midifile/craigsapp-midifile-9ee5898/include*
- **L** : Opción que permite añadir la biblioteca con la que se desea trabajar, en este caso ubicada en */home/usuario/Escritorio/midifile/craigsapp-midifile-9ee5898/lib –lmidifile*
- **pthread** : Permite la correcta ejecución de las hebras implementas.

Antes de ejecutar el código, se debe de instalar el sintetizador software encargado de la reproducción de los mensajes musicales que contiene el audio, junto con el archivo especial soundfont. El elegido para este trabajo ha sido fluidsynth, que se puede instalar escribiendo en la consola el siguiente comando:

sudo apt-get install fluidsynth

Cuando el sintetizador ha sido instalado, se lanza en un terminal nuevo con el siguiente commando:

fluidsynth -a alsa -m alsa_seq -l -n -s -o "shell.port=9800" /usr/share/sounds/sf2 /FluidR3_GM.sf2

- **a alsa**: Indica los drivers de audio a utilizar.
- **m alsa_seq**: Nombre de los drivers MIDI a usar.
- **l**: Para no conectar con un servidor LASH.
- **n**: No crea un driver para leer eventos MIDI de entrada.
- **s**: Arranca fluidsynth como un servidor.
- **o “*shell.port=9800”*:** Ajuste para indicar el puerto por el que se reciben los mensajes de audio.
- ***/usr/share/sounds/sf2 /FluidR3_GM.sf2***: Indica la localización del archivo soundfont, que contendrá la información necesaria para generar los sonidos de los instrumentos que aparecen en el archivo de audio MIDI.

Una vez que el programa ha sido compilado de manera correcta, y lanzado el sintetizador, se pasa a su ejecución con el siguiente comando:

./midi /home/usuario/Escritorio/Harry_Potter.mid

El comando está formado por el ejecutable *midi*, creado anteriormente en la compilación del código, junto con el directorio en el que se almacena el archivo de audio MIDI a reproducir, en este caso se trata de la canción de la famosa película de Harry Potter.

A continuación, en la figura 21 se puede observar el sintetizador software lanzado, esperando a recibir mensajes MIDI por parte del hilo reproducir implementado en el programa. Mientras que en la figura 22 se ve como el programa envía notas al sintetizador para que las reproduzca.

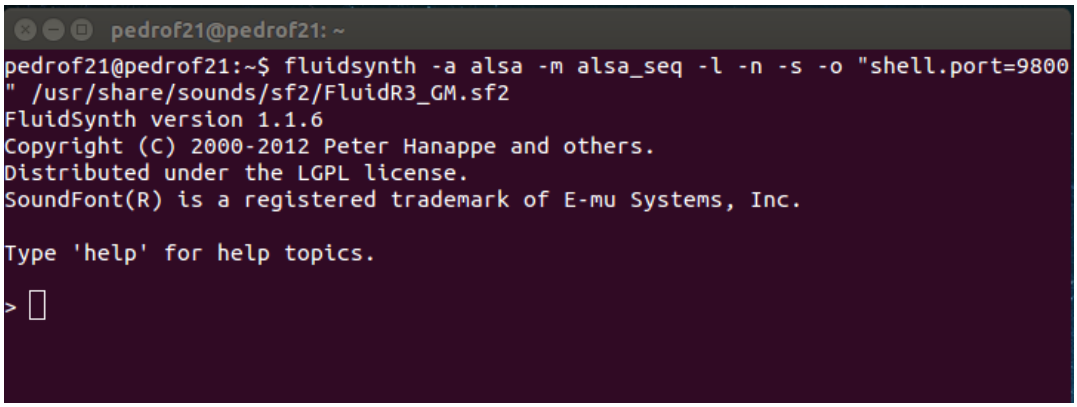

Figura 21. Sintetizador lanzado

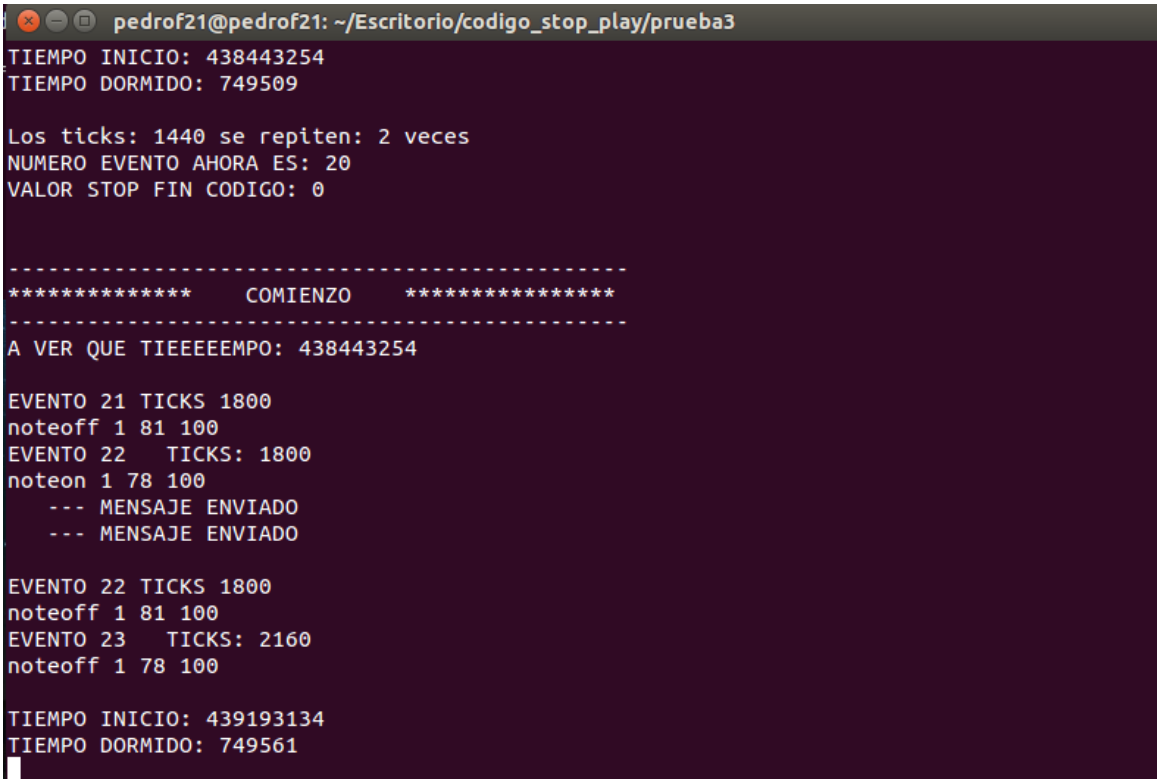

Figura 22. Programa enviando notas MIDI

7.2 ANEXO II. DOCUMENTACIÓN TÉCNICA

En este apartado aparece todo el código que se ha implementado para lograr los objetivos marcados. Se detallan los hilos, variables, métodos y estructuras que se han desarrollado.

La información aparece ordenada por página principal, estructuras, y el contenido del archivo.

Instrumento Virtual 1.0

Sintetizador

- **Página principal**
- **Estructuras de Datos**
- **Archivos**

Instrumento Virtual Documentación

Generado por doxygen 1.8.11

Estructura de datos

A continuación se muestra las estructuras que se han desarrollado junto con una breve explicación. También se muestra un índice de estructura de datos y los campos que se han utilizado en cada una de ellas.

Estructura de datos

Lista de estructuras con una breve descripción:

cdatos_thread	Estructura encargada de almacenar el archivo de audio MIDI
cestructuraevento	Estructura para almacenar mensajes del archivo MIDI
cestructurahilotempos	Estructura para almacenar tempos del fichero .txt
cestructuraplay	Estructura para almacenar beats, eventos y tiempo para dormir
cestructurareproductor	Estructura para almacenar tiempos y cadenas de texto
cestructurastop	Estructura para almacenar el valor del evento STOP
cestructuraticks	Estructura para almacenar ticks del archivo MIDI
cestructuratomarbeat	Estructura para almacenar el beat a elegir
cmiestructura	Estructura para almacenar tempos
ctcp_client	Clase Cliente TCP

Índice de estructura de datos

D

Datos_thread

E

Estructuraevento
Estructurahilotempos
Estructuraplay
Estructurareproductor
Estructurastop
Estructuraticks
Estructuratomarbeat

M

Miestructura

T

Tcp_client

Campos de datos

Todo

Lista de todos los campos de estructuras y uniones con enlaces a las estructuras/uniones a las que pertenecen:

- argcc : **datos_thread**
- argvv : **datos_thread**
- beat_es : **estructuraplay**
- cambio_evento_es : **estructuraevento**
- conn() : **tcp_client**
- dormir : **estructuraplay**
- envion_es : **estructurareproductor**
- envion_siguiente_es : **estructurareproductor**
- evente_es : **estructuraevento**
- evento_es : **estructuraplay**
- receive() : **tcp_client**
- send_data() : **tcp_client**
- stop_es : **estructurastop**
- tcp_client() : **tcp_client**
- tempo_nuevo_es : **miestructura**
- tempo_viejo_es : **miestructura**
- tfin_signal_es : **estructurareproductor**
- ticks_actual_es : **estructuraticks**
- ticks_siguiente_es : **estructuraticks**
- ticksac_tempoes : **estructurahilotempos**
- tinicio_es : **estructurareproductor**
- tomar_beat_es : **estructuratomarbeat**

Funciones

- conn() : **tcp_client**
- receive() : **tcp_client**
- send_data() : **tcp_client**
- tcp_client() : **tcp_client**

Variables

- argcc : **datos_thread**
- argvv : **datos_thread**
- beat_es : **estructuraplay**
- cambio_evento_es : **estructuraevento**
- dormir : **estructuraplay**
- envion_es : **estructurareproductor**
- envion_siguiente_es : **estructurareproductor**
- evente_es : **estructuraevento**
- evento_es : **estructuraplay**
- stop_es : **estructurastop**
- tempo_nuevo_es : **miestructura**
- tempo_viejo_es : **miestructura**
- tfin_signal_es : **estructurareproductor**
- ticks_actual_es : **estructuraticks**
- ticks_siguiente_es : **estructuraticks**
- ticksac_tempoes : **estructurahilotempos**
- tinicio_es : **estructurareproductor**
- tomar_beat_es : **estructuratomarbeat**

Referencia de la Estructura datos_thread

Estructura encargada de almacenar el archivo de audio MIDI.

Campos de datos

int	**argcc**
char **	**argvv**

Descripción detallada

Estructura encargada de almacenar el archivo de audio MIDI.

Los campos presentes son los encargados de almacenar la información que se obtiene del audio para posteriormente pasarla al hilo reproducir y que este trabaje a partir de esta información.

Documentación de los campos

int argcc

char argvv**

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura estructuraevento

Estructura para almacenar mensajes del archivo MIDI.

Campos de datos

int	**cambio_evento_es**
int	**evente_es**

Descripción detallada

Estructura para almacenar mensajes del archivo MIDI.

Los campos cambio_evento_es y evente_es son los encargados de almacenar el evento musical por el que va la reproducción del audio. Para cuando se produzcan saltos en estos que se pueda recuperar el valor actual.

Documentación de los campos

int cambio_evento_es

int evente_es

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura estructurahilotempos

Estructura para almacenar tempos del fichero .txt.

Campos de datos

int	**ticksac_tempoes**

Descripción detallada

Estructura para almacenar tempos del fichero .txt.

Parámetros

ticksac_tempo_es	Se encarga de almacenar el valor de los ticks del evento que se está reproduciendo.

Documentación de los campos

int ticksac_tempoes

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura estructuraplay

Estructura para almacenar beats, eventos y tiempo para dormir.

Campos de datos

float	**beat_es** [1000]
float	**evento_es** [1000]
float	**dormir** [1000]

Descripción detallada

Estructura para almacenar beats, eventos y tiempo para dormir.

Parámetros

beat_es	Almacena los tiempos de beats en los que se compone la pista de audio.
evento_es	Almacena el valor del primer evento de cada tiempo de beat.
dormir	Presenta el tiempo a dormir en cada tiempo de beat.

Documentación de los campos

float beat_es[1000]

float dormir[1000]

float evento_es[1000]

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura estructurareproductor

Estructura para almacenar tiempos y cadenas de texto.

Campos de datos

int	**tinicio_es**
int	**tfin_signal_es**
string	**envion_es**
string	**envion_siguiente_es**

Descripción detallada

Estructura para almacenar tiempos y cadenas de texto.

Los campos tinicio_es y tfin_signal_es almacenan valores temporales para sincronizar la reproducción del audio mientras que los campos envion_es y envion_siguiente_es presentan una cadena de texto con la nota a reproducir y la siguiente.

Documentación de los campos

string envion_es

string envion_siguiente_es

int tfin_signal_es

int tinicio_es

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura estructurastop

Estructura para almacenar el valor del evento STOP.

Campos de datos

float **stop_es**

Descripción detallada

Estructura para almacenar el valor del evento STOP.

Parámetros

stop_es Almacena el estado de la reproducción del audio.

Documentación de los campos

float stop_es

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura estructuraticks

Estructura para almacenar ticks del archivo MIDI.

Campos de datos

int	**ticks_actual_es**
int	**ticks_siguiente_es**

Descripción detallada

Estructura para almacenar ticks del archivo MIDI.

Los campos de datos utilizados, ticks_actual_es y ticks_siguiente_es se utilizan para medir la duración de una nota. Cuando ticks_siguiente_es presenta un valor distinto a ticks_actual_es, la diferencia de estos dos será el tiempo que el hilo reproducir esté durmiendo.

Documentación de los campos

int ticks_actual_es

int ticks_siguiente_es

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura estructuratomarbeat

Estructura para almacenar el beat a elegir.

Campos de datos

float	**tomar_beat_es**

Descripción detallada

Estructura para almacenar el beat a elegir.

Parámetros

tomar_beat_es	Almacena el valor de beat elegido recibido en el evento play.

Documentación de los campos

float tomar_beat_es

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Estructura miestructura

Estructura para almacenar tempos.

Campos de datos

float	**tempo_viejo_es**
float	**tempo_nuevo_es**

Descripción detallada

Estructura para almacenar tempos.

Parámetros

tempo_viejo_es	Se encarga de almacenar el valor del tempo con el que este reproduciéndose el archivo de audio.
tempo_nuevo_es	Cuando se recibe un tempo nuevo, se realiza el cálculo para adaptar la velocidad de reproducción a dicho tempo, haciendo uso del tempo viejo.

Documentación de los campos

float tempo_nuevo_es

float tempo_viejo_es

La documentación para esta estructura fue generada a partir del siguiente fichero:

- **main.cpp**

Referencia de la Clase tcp_client

Clase Cliente TCP.

Métodos públicos

	tcp_client ()
bool	conn (string, int)
bool	send_data (string data)
string	receive (int)

Descripción detallada

Clase Cliente TCP.

Cliente encargado de enviar los mensajes musicales (notas), al sintetizador encargado de reproducirlas.

Documentación del constructor y destructor

tcp_client ( )

```
53 {
54  sock = -1;
55  port = 0;
56  address = "";
57 }
```

Documentación de las funciones miembro

bool conn (string address,
int port
)

Conecta con un host en un determiando puerto.

```
63 {
64  //Crea un socket si este no esta creado aun
65  if(sock == -1)
66  {
67  //Crea el socket
68  sock = socket(AF_INET , SOCK_STREAM , 0);
69  if (sock == -1)
```

```
70  {
71  perror("No se puede crear el socket");
72  }
73
74  cout<<"Socket creado\n";
75  }
76  else { /* OK , nada */ }
77
78  //Configura la estructura de direccion
79  if(inet_addr(address.c_str()) == -1)
80  {
81  struct hostent *he;
82  struct in_addr **addr_list;
83
84  //Resuelve el hostname, no es una direccion ip
85  if ( (he = gethostbyname( address.c_str() ) ) == NULL)
86  {
87  //gethostbyname fallido
88  herror("gethostbyname");
89  cout<<"Fallo al resolver el hostname\n";
90
91  return false;
92  }
93
94  addr_list = (struct in_addr **) he->h_addr_list;
95
96  for(int i = 0; addr_list[i] != NULL; i++)
97  {
98  server.sin_addr = *addr_list[i];
99
100 cout<<address<<" resuelto a "<<inet_ntoa(*addr_list[i])<<endl;
101
102 break;
103 }
104 }
105
106 //Direccion ip sin formato
107 else
108 {
109 server.sin_addr.s_addr = inet_addr( address.c_str() );
110 }
111
112 server.sin_family = AF_INET;
113 server.sin_port = htons( port );
114
115 //Conecta a servidor remoto
116 if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
117 {
118 perror("Conexion fallida. Error");
119 return 1;
120 }
121
122 cout<<"Conectado\n";
123 return true;
124 }
```

string receive (int)
bool send_data (string data)

Envia datos al host conectado

```
130 {
131  //Envia datos
132  if( send(sock , data.c_str() , strlen( data.c_str() ) , 0) < 0)
133  {
134  perror("Envio fallido : ");
135  return false;
136  }
137  cout<<" --- MENSAJE ENVIADO\n";
138 
139  return true;
140 }
```

La documentación para esta clase fue generada a partir del siguiente fichero:

- **main.cpp**

Archivos

Globales

Todo

Lista de todas las funciones, variables, 'defines', enumeraciones y 'typedefs' con enlaces a los ficheros a los que corresponden:

- estructuraEvento : **main.cpp**
- estructuraHilotempos : **main.cpp**
- estructuraPlay : **main.cpp**
- estructuraReproductor : **main.cpp**
- estructuraStop : **main.cpp**
- estructuraTicks : **main.cpp**
- estructuraTomarBeat : **main.cpp**
- hilo_reproducir() : **main.cpp**
- hilo_tempos() : **main.cpp**
- main() : **main.cpp**
- miEstructura : **main.cpp**
- puntero : **main.cpp**
- puntero1 : **main.cpp**
- puntero2 : **main.cpp**
- puntero3 : **main.cpp**
- puntero4 : **main.cpp**
- puntero5 : **main.cpp**
- puntero6 : **main.cpp**
- puntero7 : **main.cpp**
- sig_handler() : **main.cpp**

Funciones

- hilo_reproducir() : **main.cpp**
- hilo_tempos() : **main.cpp**
- main() : **main.cpp**
- sig_handler() : **main.cpp**

Variables

- estructuraEvento : **main.cpp**
- estructuraHilotempos : **main.cpp**
- estructuraPlay : **main.cpp**
- estructuraReproductor : **main.cpp**
- estructuraStop : **main.cpp**
- estructuraTicks : **main.cpp**
- estructuraTomarBeat : **main.cpp**
- miEstructura : **main.cpp**
- puntero : **main.cpp**
- puntero1 : **main.cpp**
- puntero2 : **main.cpp**
- puntero3 : **main.cpp**
- puntero4 : **main.cpp**
- puntero5 : **main.cpp**
- puntero6 : **main.cpp**
- puntero7 : **main.cpp**

Lista de archivos

Cabeceras del Archivo main.cpp

```
#include "MidiFile.h"
#include "Options.h"
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <string.h>
#include <string>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <sstream>
#include <math.h>
#include <stdlib.h>
#include <ctime>
#include <cstdio>
#include <time.h>
#include <sys/time.h>
#include <pthread.h>
#include <signal.h>
#include <fstream>
```

Estructuras de datos

class	**tcp_client** Clase Cliente TCP.
struct	**estructuraticks** Estructura para almacenar ticks del archivo MIDI.
struct	**estructuraevento** Estructura para almacenar mensajes del archivo MIDI.
struct	**miestructura** Estructura para almacenar tempos.
struct	**estructurareproductor** Estructura para almacenar tiempos y cadenas de texto.
struct	**estructurahilotempos** Estructura para almacenar tempos del fichero .txt.
struct	**estructurastop** Estructura para almacenar el valor del evento STOP.
struct	**estructuraplay** Estructura para almacenar beats, eventos y tiempo para dormir.
struct	**estructuratomarbeat** Estructura para almacenar el beat a elegir.
struct	**datos_thread** Estructura encargada de almacenar el archivo de audio MIDI.

Variables

struct estructuraticks	estructuraTicks
struct estructuraticks *	puntero
struct estructuraevento	estructuraEvento
struct estructuraevento *	puntero1
struct miestructura	miEstructura
struct miestructura *	puntero2
struct estructurareproductor	estructuraReproductor
struct estructurareproductor *	puntero3
struct estructurahilotempos	estructuraHilotempos
struct estructurahilotempos *	puntero4
struct estructurastop	estructuraStop
struct estructurastop *	puntero5
struct estructuraplay	estructuraPlay
struct estructuraplay *	puntero6
struct estructuratomarbeat	estructuraTomarBeat
struct estructuratomarbeat *	puntero7

void sig_handler (int signo)

Función SIG_HANDLER.

Función encargada de direccionar el código en función del evento que haya recibido. Esta función es llamada cuando un determinado evento llega al servidor colocado en el hilo principal main. De modo que detiene la reproducción de archivo de audio MIDI para que este método indique que parte del código del hilo_reproducir se ha de ejecutar.

void * hilo_tempos (void * arg)

Hebra encargada del envío de eventos.

Este hilo lee de un archivo texto los eventos que han de ser enviados cada cierto tiempo. La información aparece ordenada en cuatro columnas, en la primera se indica el orden en el que los eventos van a ser enviados, la segunda indica el instante de tiempo en el que los eventos van a ser enviados, la tercera indica el tipo de evento a enviar (tempo, stop o play) y la cuarta indica el beat en el que se tiene que iniciar la reproducción cuando se reciba un evento play.

void * hilo_reproducir (void * arg)

Hebra encargada de la reproducción del archivo de audio MIDI.

Este hilo realiza varias acciones:

- Utiliza la biblioteca MidiFile para obtener toda la información del archivo de audio.
- Cambio de formato en los mensajes musicales obtenidos.
- Cliente TCP para enviar estos mensajes al sintetizador.
- Calculo del nuevo tempo recibido.
- Detiene la reproducción del audio.
- Continúa con la reproducción del audio.

int main (int argc, char** argv)

Hebra principal (MAIN)

Hilo encargado de lanzar las hebras secundarias, hilo_reproducir e hilo_tempos. Aparece un servidor TCP encargado de recibir los eventos que son enviados a través de red.

Documentación de las variables

struct estructuraevento **estructuraEvento**

struct estructurahilotempos **estructuraHilotempos**

struct estructuraplay **estructuraPlay**

struct estructurareproductor **estructuraReproductor**

struct estructurastop **estructuraStop**

struct estructuraticks **estructuraTicks**

struct estructuratomarbeat **estructuraTomarBeat**

struct miestructura **miEstructura**

struct estructuraticks * **puntero**

struct estructuraevento * **puntero1**

struct miestructura * **puntero2**

struct estructurareproductor * **puntero3**

struct estructurahilotempos * **puntero4**

struct estructurastop * **puntero5**

struct estructuraplay * **puntero6**

struct estructuratomarbeat * **puntero7**

7.3 ANEXO III. ÍNDICE DE FIGURAS

- Figura 1. Estructura binaria de un mensaje MIDI
- Figura 2. Telégrafo musical
- Figura 3. Telarmoni
- Figura 4. Sonido gráfico
- Figura 5. Teclado Novachord
- Figura 6. Sintetizador moog
- Figura 7. Sintetizador prophet-5
- Figura 8. Qsynth (GUI Fluidsynth)
- Figura 9. Estructura mensaje OSC
- Figura 10. Mensajes OSC
- Figura 11. Cadena OSC
- Figura 12. Matriz de bytes
- Figura 13. Información obtenida por biblioteca MidiFile
- Figura 14. Hebras del programa
- Figura 15. Diagrama de flujo de recepción de eventos
- Figura 16. Envío de eventos MIDI al sintetizador
- Figura 17. Diagrama de envío de mensajes MIDI
- Figura 18. Diagrama de recepción de eventos en el hilo reproducir
- Figura 19. Sincronización temporal de hebras

7.4 ANEXO IV. ÍNDICE DE TABLAS

- Tabla 1. Cuadro sinóptico de los mensajes MIDI
- Tabla 2. Tabla de notas MIDI
- Tabla 3. Lista de instrumentos GM (General MIDI)
- Tabla 4. Mensaje de sistema exclusivo
- Tabla 5. Tipos de datos estándar
- Tabla 6. Tabla de eventos para pruebas

8. REFERENCIAS BIBLIOGRÁFICAS

1.) Antonio Garrido Carrillo. Fundamentos de programación en C++. Madrid: Delta, 2006.

2.) Bjarne Stroustrup. El lenguaje de programación C++. Madrid: Addison Wesley, 2002.

3.) En Ardour, 2016: https://ardour.org/

4.) En audio digital y MIDI, 2015: http://www.ccapitalia.net/reso/articulos/audiodigital/

5.) En blitter, 2016: http://www.blitter.com/~russtopia/MIDI/~jglatt/tech/midifile/ppqn.htm

6.) En C++, 2016: http://www.cplusplus.com/

7.) En Doxygen, 2016: https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html

8.) En Fluidsynth, 2016: http://www.fluidsynth.org/

9.) En infor.uva, 2016: http://www.infor.uva.es/~benja/creacion-hilos.html

10.) En Linux, 2016: http://man7.org/linux/man-pages/man2/kill.2.html

11.) En Linux, 2016: http://linux.die.net/man/1/kill

12.) En MidiFile, 2016: http://midifile.sapp.org/

13.) En opensoundcontrol, 2016: http://opensoundcontrol.org/spec-1_0

14.) En Renoise, 2016: https://www.renoise.com/

15.) En taringa, 2016: http://www.taringa.net/posts/info/3510820/La-historia-del-sintetizador.html

16.) En TiMidity, 2016: http://timidity.sourceforge.net/

17.) En Wikipedia, 2016: https://es.wikipedia.org/wiki/MIDI

18.) En Wikipedia, 2016: https://es.wikipedia.org/wiki/Sintetizador

19.) En Wikipedia, 2016: https://es.wikipedia.org/wiki/OpenSound_Control

20.) En tutorialspoint, 2016: http://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm

21.) Enrique Hernández Orallo. Programación C++. Madrid: Paraninfo, 1995.

22.) F.J. Rodriguez-Serrano, J.J. Carabias-Orti, P. Vera-Candeas and D. Martinez-Munoz, 2016. Tempo Driven Audio-to-Score Alignment using Spectral Decomposition and Online Dynamic Time Warping. ACM Trans. Embedd. Comput. Syst. 9, 4, Article 39 (February 2016), 20 pages.

23.) José Manuel Muñoz Frías. Fundamentos de programación utilizando el lenguaje C. Universidad Pontificia Comillas, 2006.

24.) Luis Joyanes Aguilar. Programación en C: metodología, algoritmos y estructuras de datos. Madrid: McGraw-Hill, cop 2005.
25.) Stephen Prata. C++ primer plus. Upper Saddle River : Addision-Wesley Professional, 2012.
26.) Sistema de acompañamiento musical: http://music.informatics.indiana.edu/~craphael/papers/chance.pdf
27.) Video antescofo: https://www.youtube.com/watch?v=YkMGtpcAA04
28.) Software 2015, Pure Data: https://puredata.info/