Tercera tarea programada

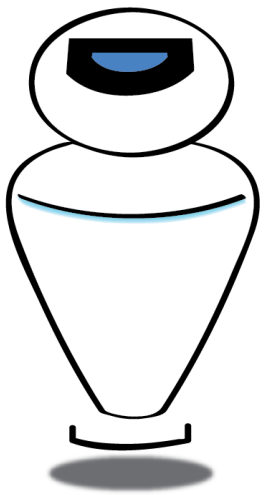
LISP

Reproductor de mp3

Fernanda Fernández

Miguel Ángel Gutiérrez

Yesenia Montiel



Contenido

Tarea Programada 3: LISP
Reprodutor de mp3

Introducción

Descripción del problema

El objetivo de esta tarea es familiarizarse con el desarrollo de aplicaciones usando el lenguaje Lisp, mediante la creación de una base de datos de información de archivos de música en mp3.

La meta-información de los archivos mp3 está codificada usando un formato llamado ID3. Deberán escribir funciones de Lisp que permitan leer un archivo mp3, y decodificar la información presente en el header de ID3.

Para decodificar un archivo binario como un mp3 deberán escribir rutinas para poder leer archivos binarios.

Posteriormente, deberán crear una base de datos que se almacenará en memoria, la cual tendrá una estructura para poder almacenar la información de los mp3 que se encuentra en un header ID3.

La estructura de la base de datos la pueden definir ustedes, pero la idea es que se basen en el formato ID3.

Deberán tener una función que permita especificar un directorio de la computadora, y posteriormente que dicha función lea todos los mp3 de dicho directorio, extraiga la información de los headers ID3, y almacene esa información en la base de datos de memoria.

Se deberán crear funciones para poder hacer las siguientes consultas sobre la base de datos de música:

- Obtener todas las canciones de un determinado artista
- Obtener todas las canciones de un determinado género
- Obtener todas las canciones de un determinado álbum

Tarea Programada 3: LISP
Reprodutor de mp3

Diseño del programa

Funciones:

• (**lee-id3** "archivo-mp3"):
Abre el archivo que recibe como parámetro como binario y lo interpreta de la siguiente forma: los primeros tres bytes son de la etiqueta ID3, el siguiente byte es la versión, luego la revisión y el flag, y los siguientes cuatro bytes son el largo de la etiqueta. Lee los bytes de la etiqueta y convierte cada byte a char y lo concatena a una variable string con el contenido de la etiqueta.
• (**insert** objeto-mp3)
Inserta un objeto mp3 con los datos de una canción en la estructura de datos que almacena las canciones (hash-table).
• (**consulta-genero** "genero")
Imprime los objetos en el hash-table que cumplen con el slot género igual al parámetro.
• (**consulta-artista** "artista")
Imprime los objetos en el hash-table que cumplen con el slot artista igual al parámetro.
• (**consulta-album** "album")
Imprime los objetos en el hash-table que cumplen con el slot album igual al parámetro.
• (**carga-directorio** "directorio")
Itera sobre los archivos mp3 del directorio llamando a ¿¡?¡?¡?¡ que saca la información del ID3 usando la función (lee-id3 "archivo") y crea los objetos para insertarlos en el hash table.
• (**revisar string** lista)
Revisa si dentro del string generado de la lectura del archivo mp3 existe etiquetas si las hay llama a replaceString (); la lista contiene las etiquitas estándar del ID3 v3
• (**replaceString** parte stringOriginal remplazo)
Recibe la parte del string a sustituir el string original y un comodin para remplazar las etiquetas, las etiquetas que se necesitas las remplaza por "$ numero $" eso hace que me marque la información para después recuperarla
• (**split-by-$** string)
Recibe un string y le hace un Split por $ lo cual convierte en una lista los valores necesarios
• (**crear_mp3** lista)
Recibe la lista generada por el (split-by-$ string) el cual separa el contenido necesario para llamar a Insert para llenar la tabla hash.

Tarea Programada 3: LISP
Reprodutor de mp3

Decisiones tomadas

Base de Datos

La base de datos se carga en tiempo de ejecución mediante la función (carga-directorio "directorio").

Consultas

Después de cargar la base de datos en memoria el usuario puede utilizar las siguientes funciones para realizar las consultas:

- (consulta-artista "nombre-artista"): obtener las canciones de un determinado artista

- (consulta-genero "nombre-genero"): obtener las canciones de un determinado género

- (consulta-album "nombre-album"): obtener las canciones de un determinado álbum

Estructuras de Datos

El programa es diseñado según los requerimientos de las 3 consultas base. Para cada una de ellas se realizaron diferentes funciones que permiten consultar los datos solicitados por el usuario, para ello se requirió de estructuración de una clase principal mp3-file. Esta adapta cuatro atributos importantes, las cuales son: nombre, artista, género y álbum; en ellos se almacenaran los datos principales de las consultas requeridas y que serán manipuladas en una estructura de datos de tabla hash. Veamos la definición de la clase principal:

```
(defclass mp3-file ()
((nombre :initarg :nombre :initform "")
(artista :initarg :artista :initform "")
(genero :initarg :genero :initform "")
(album :initarg :album :initform "")
))
```

A partir de la definición anterior, se requiere de la estructura de almacenamiento de la tabla hash (*ht*) para almacenar los atributos del objeto, de esta forma llenada la tabla y poder realizar funciones de mapeo para iterar el contenido del mismo, esto se da el por medio de la siguiente instrucción:

```
(maphash #'(lambda (k v) (función v parametro)) *ht*))
```

Luego por medio de instrucciones de comparación de llaves la función devuelve el valor de la llave encontrada.

La tabla hash fue creada por medio del a siguiente instrucción:

```
(defparameter *ht* (make-hash-table))
```

Y su método para inserción de llaves y valores a la tabla es la siguiente instrucción:

```
(defun insert (a-mp3) (setf (gethash(slot-value a-mp3 'nombre) *ht*) a-
mp3))
```

Lógica del programa

La clase mp3-file () antes mencionada tendrá un conjunto de funciones de complemento para realizar las tres consultas base de carga de datos: consulta-artista, consulta-genero, consulta-album.

Primero carga-directorio(directorio), que tendrá la función de solicitar un directorio donde se encontrará los archivos.mp3, esta función está conformada por un loop que itera sobre el directorio solicitado, esta lee todos los archivos de extensión .mp3 y los concatena el directorio dado con el nombre del archivo y su extensión .mp3 Ejemplo: directorio = “/home/fer/Escritorio”,

Veamos la función en las siguientes líneas de código.

```
(defun carga-directorio(directorio)
(loop for f in (directory (concatenate 'string directorio "/*.mp3"))
  collect (lee-id3 f)))
```

El resultado de la concatenación del string es parametrizado sobre la función de lee-id3 a-mp3, de la cual este abre el archivo en formato binario con el directorio que se generó con la función cargadirectorio.

Una vez leído el archivo en binario, la función inicializa tres variables: byte, tag-size y tags.
Luego empieza a leer los tres primeros bytes, consta del encabezado del ID3 de archivo, por ejemplo: IDC; los siguientes tres bytes leídos son la versión, la revisión y las banderas. Por ejemplo: IDC100; por consiguiente continua la lectura de los 4 bytes posteriores que representan la longitud de la etiqueta.

Y se calcula la siguiente manera:

```
(setf *tag-size* (+(*(read-byte *archivo*) 16777216) (*(read-byte *archivo*) 65536)
(*(read-byte
*archivo*) 256) (read-byte *archivo*))) ;4 bytes de tamaño de tags.
```

Posteriormente se realiza un ciclo de lectura de los 5000 bytes posteriores, pero va a ir concatenando los bytes que son necesarios, como etiquetas, nombre, álbum, genero, artista, edición, entre otras, por otra parte también excluirá muchos de los códigos innecesarios que posee el ID3 del archivo.

Veamos la función en las siguientes líneas de código.

```
(defun lee-id3 (a-mp3)
 (setf *archivo* (open a-mp3 :element-type 'unsigned-byte))
 (setf *byte* 0)
 (setf *tag-size* 0)
 (setf *tags* "")
 (code-char (read-byte *archivo*));I
 (code-char (read-byte *archivo*));D
 (code-char (read-byte *archivo*));C
 (read-byte *archivo*) ;major version
 (read-byte *archivo*) ;revisión
(read-byte *archivo*) ;flags
(setf *tag-size* (+(*(read-byte *archivo*) 16777216) (*(read-byte *archivo*) 65536)
(*(read-byte
*archivo*) 256) (read-byte *archivo*))) ;4 bytes de tamaño de tags
 (dotimes (i 5000)
  (if (and (>= (setf *byte* (read-byte *archivo*)) 0) (<= *byte* 31))
     nil
    ;(format t "~a" (code-char *x*))))
    (setf *tags* (concatenate 'string *tags* (princ-to-string(code-char *byte*))))))
```

La función revisar recibe un String y una lista. Busca en la cadena los tag si existe manda un comodín conformado por 4 caracteres en el caso que sea un tag que no se necesite el comodín seria “$$$$” y si es uno de los que se necesita se transforma en “$numero$” Por ejemplo:
Si es la etiqueta TIT2 se coloca un “01” el cual lo identifica con el nombre de la canción.
Si es la etiqueta TPE1 se coloca un “02” el cual lo identifica con el nombre del artista.
Si es la etiqueta TCON se coloca un “03” el cual lo identifica con el género
Si es la etiqueta TALB se coloca un “04” el cual lo identifica con el nombre del álbum.

```
(defun revisar(string lista)
        (loop for i in lista do(cond
```

```
                ((string-equal i  "TIT2")  (setf string (replaceString (string-downcase
                (string i)) (string-downcase string)
                "$01$")));;titulo cancion
                ((string-equal i  "TPE1")  (setf string (replaceString (string-downcase
                (string i)) (string-downcase string)
                "$02$")));;artista
((string-equal i  "TCON")  (setf string (replaceString (string-downcase (string i))
(string-downcase string)
"$03$")));;genero
((string-equal i  "TALB")  (setf string (replaceString (string-downcase (string i))
(string-downcase string)
"$04$")));;album
(t (setf string (replaceString (string-downcase (string i)) (string-downcase string)
"$$$$")))
))string)
```

Esta función llama a replaceString que realiza el proceso de cambio.
La función replaceString() recibe como parámetro la parte del string a reemplazar, el string original y el comodín de remplazo.
Al final concatena todos los String resultantes del cambio y me retorna el string resultante.

```
;;Remplaza los tags por comodines
(defun replaceString (parte stringOriginal remplazo)
 (cond ((NULL (search parte stringOriginal)) stringOriginal)
 (T(setf (subseq stringOriginal (search parte stringOriginal)
 (+ (search parte stringOriginal) (length parte))) remplazo)
 (replaceString parte stringOriginal remplazo))))
```

La función split-by-$(string) es una versión de un Split que busca el carácter $ y separa en una lista las subcadenas del string que está entre un carácter $ y otro.

```
;;Recibe un string y le hace un split sobre el caracter $
(defun split-by-$(string)
   (loop for i = 0 then (1+ j)
        as j = (position #\$ string :start i)
        collect (subseq string i j)
        while j))
```

La función crear_mp3 toma una lista generada de la función revisar y busca los comodines “01” al “04”. Cuando los encuentra toma la siguiente posición de la lista y la asigna a una variable correspondiente por ejemplo “01” corresponde a nombre de la canción el cual la asigna a Nombre.

```
(defun crear_mp3 (lista &optional (Nombre "Sin nombre") (Artista "Sin Artista")
(Genero "Sin genero")
(Album "Sin album") )
```

```
(cond
 ((eql lista '())
   (insert (make-instance 'mp3-file :nombre Nombre :artista Artista :genero Genero
:album Album)))
 ((string-equal (first lista)  "01")
   (crear_mp3 (cdr lista) (first (cdr lista)) Artista Genero Album ))
 ((string-equal (first lista)  "02")
   (crear_mp3 (rest lista) Nombre (first (rest lista))  Genero Album ))
 ((string-equal (first lista)  "03")
   (crear_mp3 (rest lista) Nombre Artista  (first (rest lista)) Album ))
 ((string-equal (first lista)  "04")
   (crear_mp3 (rest lista) Nombre Artista  Genero (first (rest lista))))
 (t
   (crear_mp3 (rest lista) Nombre Artista Genero Album ))
))
```

Una vez complementado todas las funciones anteriores, ya se puede realizar las siguientes consultas:

- consultar-genero (genero)
- consultar-artista(artista)
- consultar-abum(album)

Todas estas funciones anteriores tienen el mismo esquema lógico de operación, donde cada una de ellas, realiza un mapeo a la tabla hash, con el valor a buscar, ya sea el género, artista o el álbum. Si el valor a buscar (key) se encentra, la solicita el resto de valores de la llave encontrada.

Veamos la función en las siguientes líneas de código.

```
(defun imprimir-genero (genero)
  (maphash #'(lambda (k v) (print-gen v genero)) *ht*))

(defun imprimir-valor (valor)
  (maphash #'(lambda (k v) (print-gen v valor)) *ht*))
(defun print-valor(a-mp3 valor)
  (if (STRING-EQUAL valor (slot-value a-mp3 'valor))
    (progn
 (format t (slot-value a-mp3 'nombre))
 (format t "~%")
 (format t (slot-value a-mp3 'artista))
 (format t "~%")
 (format t (slot-value a-mp3 'album))
 (format t "~%")
 (format t (slot-value a-mp3 'genero))
 (format t "~%")
          (format t "~%")))
```

Tarea Programada 3: LISP
Reprodutor de mp3

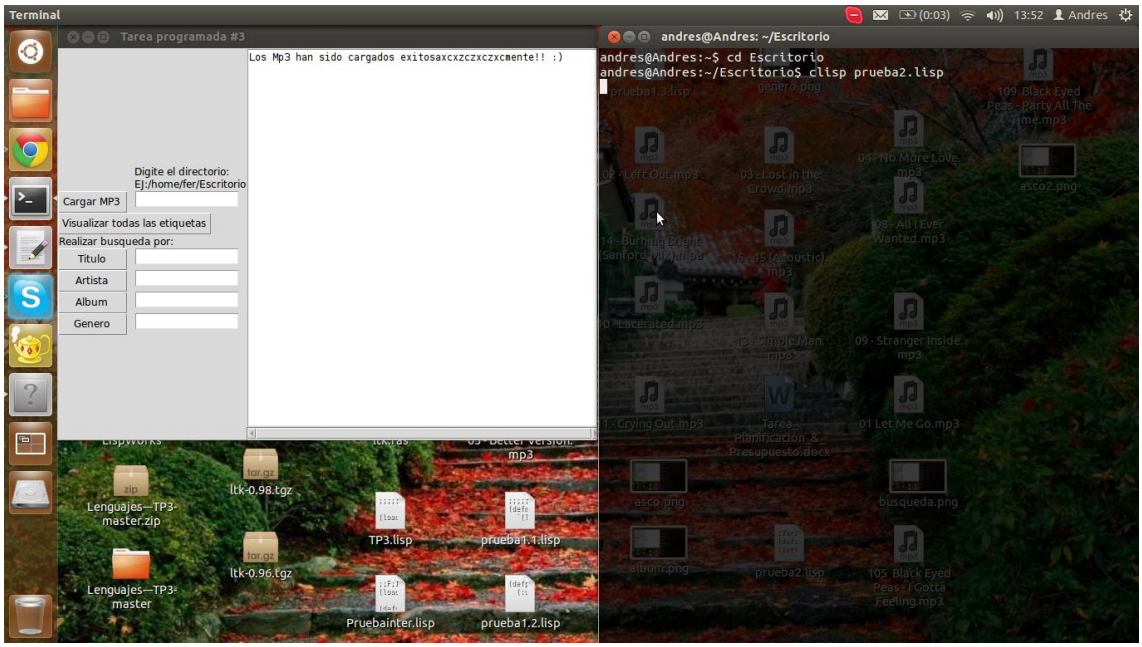
Manual de usuario

Primeramente iniciamos la terminal, digitando:

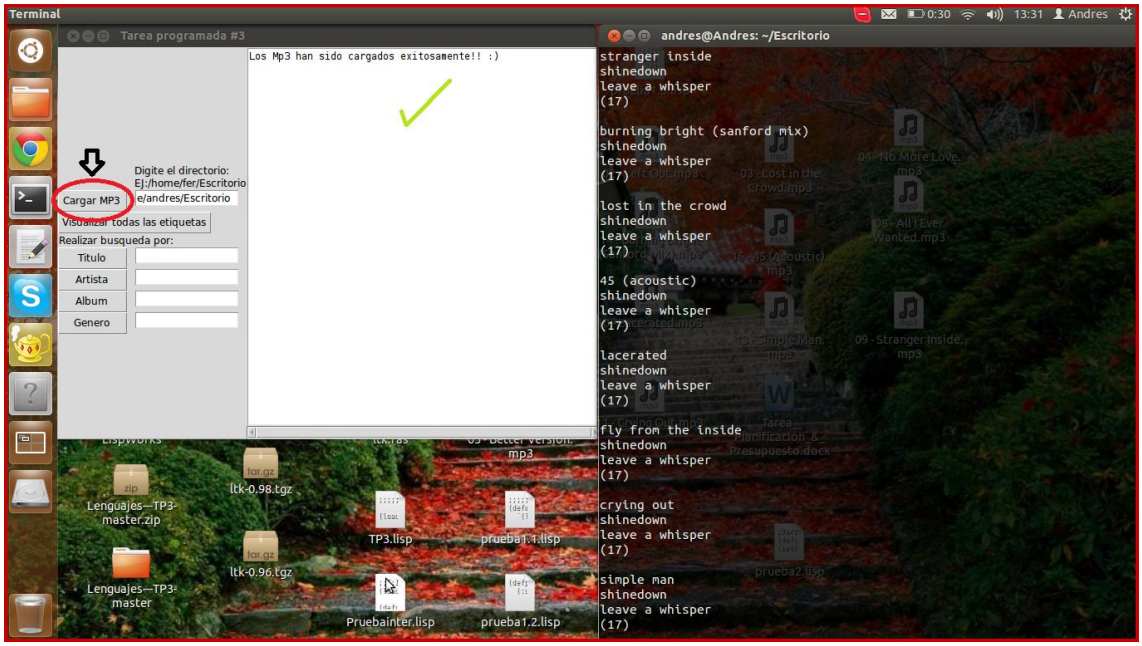
Cd Escritorio

Clisp prueba2.lisp cm se muestra en la pantalla

Se desplegará la pantalla que observamos a la izquierda donde se ejecutara nuestra programa.

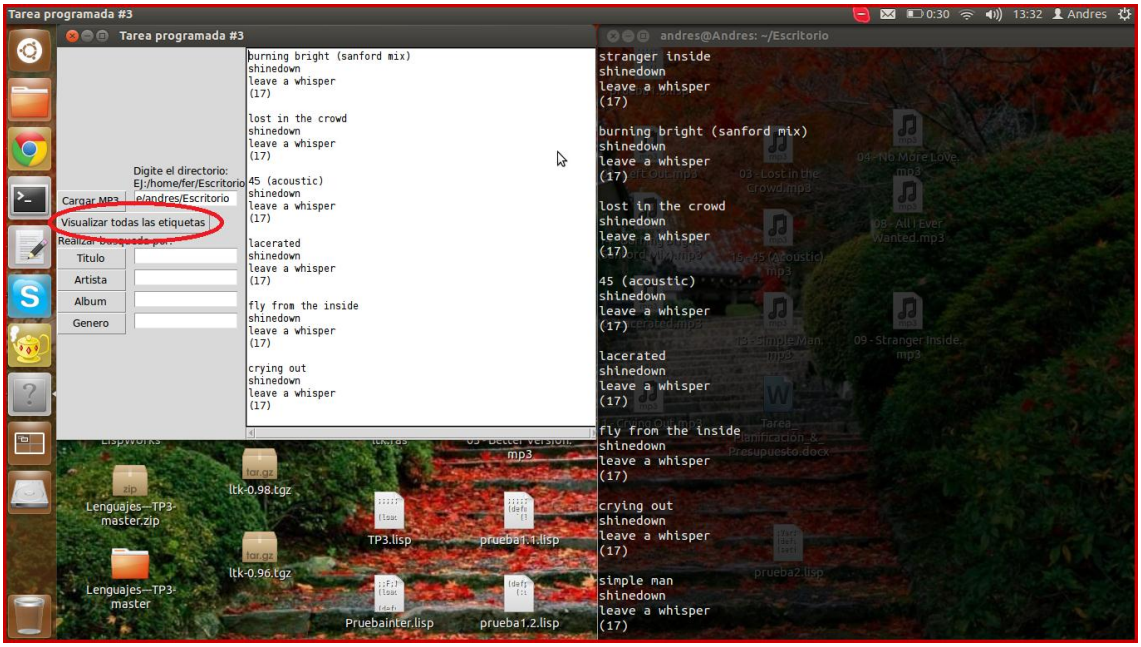


Pro seguidamente digitaremos el directorio en la que se muestra la música, con el formato que se detalla en la interfaz como lo es /home/fer/Escritorio y a la hora de clikear el botón de CARGAR MP3 podremos almacenar en nuestra base de datos todos nuestros mp3



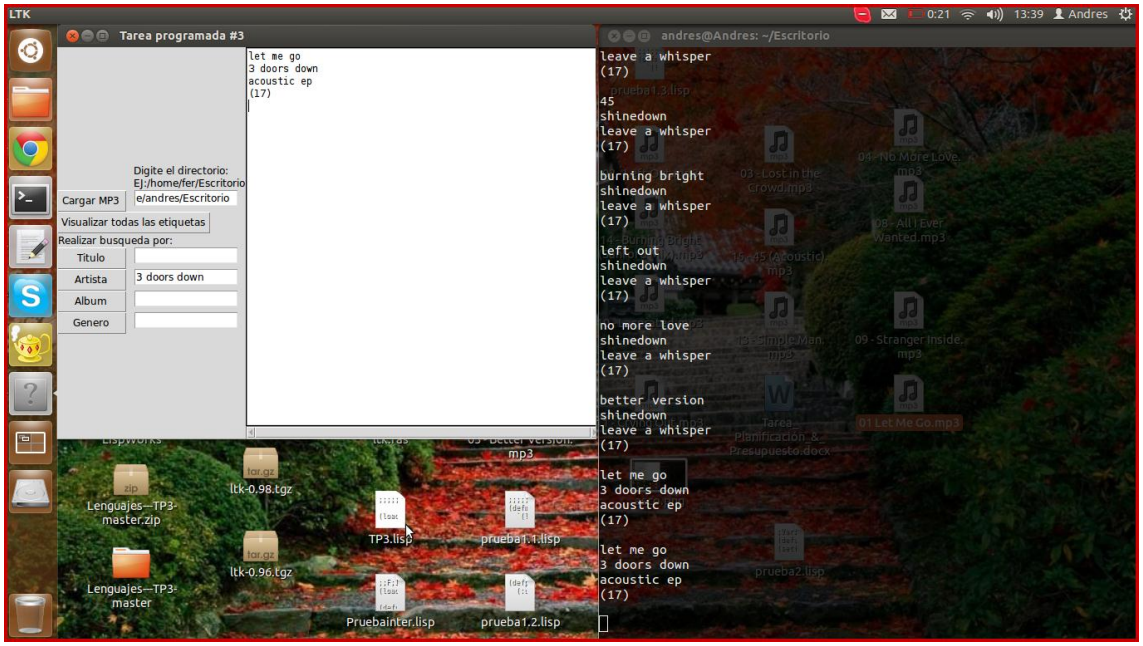
Seguidamente visualizaremos todo las etiquetas de los archivos encontrados en el directorio.

Tarea Programada 3: LISP
Reprodutor de mp3



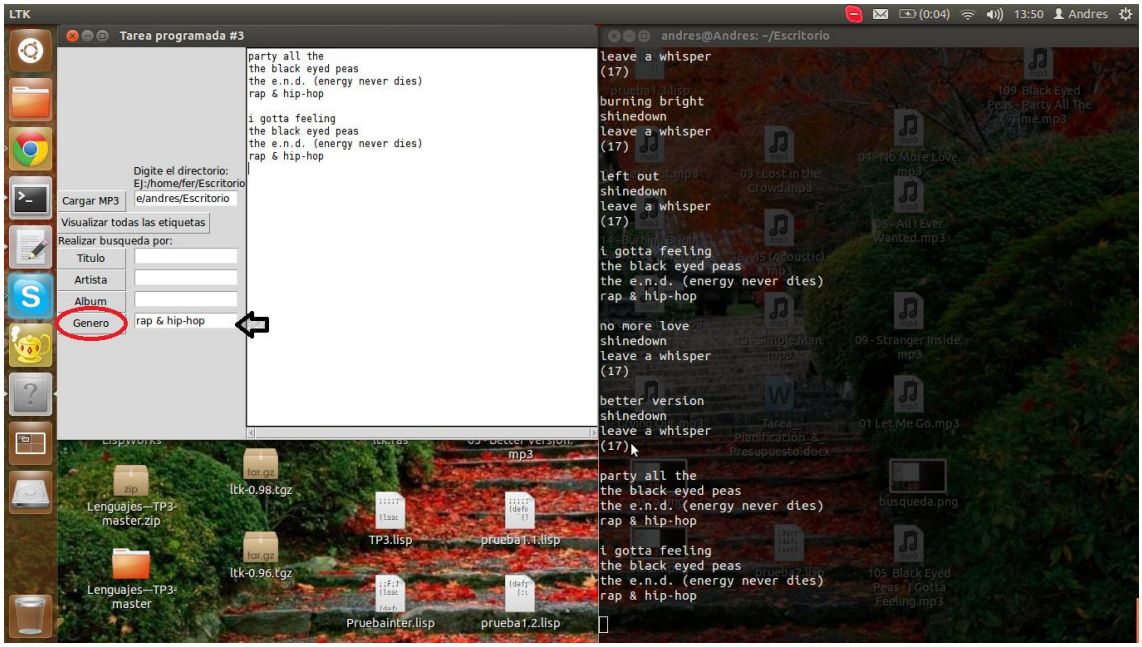
De este modo podemos observar todas las características de los mp3, su título, su artista, su álbum finalmente su género.

Adjunto los pantallazos de las búsquedas ya que cuentan con el mismo formato:

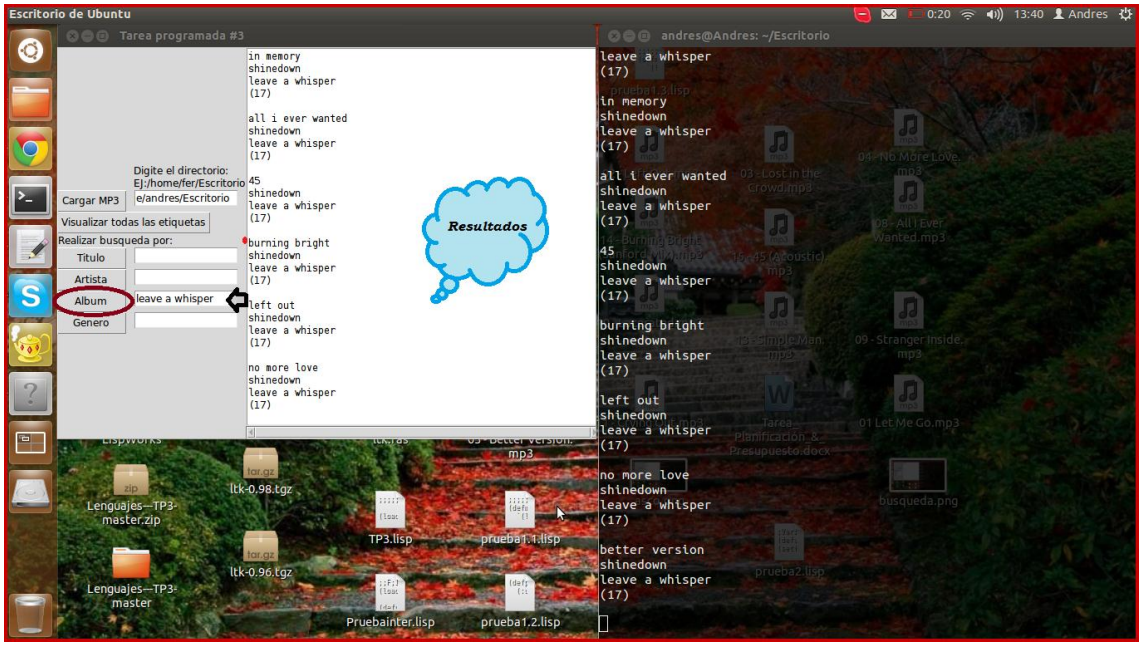


Digitar los valores en los entrys, con lo que necesite consultar y luego presiona el botón de título, artista, álbum o genero depende del valor rellenado.

Tarea Programada 3: LISP
Reprodutor de mp3



Y finalmente la búsqueda por álbum



Conclusión

Durante el desarrollo de esta tarea se aprendió mucho sobre el manejo de pdf y la información que contienen también del manejo de los directorios, además de familiarizarnos con el lenguaje lisp.

Integrantes	Investigación	Desarrollo	Documentación
FER	100%	100%	100%
JESS	100%	100%	100%
MIGUE	100%	100%	100%

Bibliografía

- stackoverflow. (2011). Recuperado el 24 de Mayo de 2011, de http://stackoverflow.com/questions/1403717/how-do-i-iterate-through-a-directory-in-common-lisp

- Apuntes de Common LISP. (s.f.). Recuperado el 23 de Mayo de 2011, de http://lear.inforg.uniovi.es/ia/Archivos/Pr%C3%A1cticas/CommonLisp.pdf

- Common Lisp the Language, 2nd Edition . (s.f.). Recuperado el 23 de Mayo de 2011, de http://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node1.html

- Seibel, P. (2009). Recuperado el 23 de Mayo de 2011, de Practical Common Lisp: http://gigamonkeys.com/book/