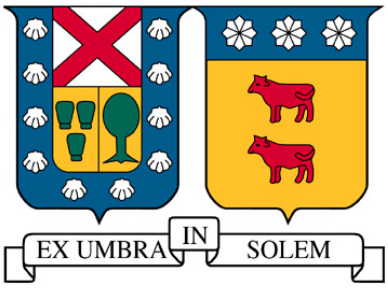

Reproductor MP3 en consola
Basado en mpg123.

jose.hidalgo@alumnos.utfsm.cl

1. Índice

2. *Introducción*

La música, quedando a opinión del lector lo que esto signifique, provoca en las personas un estado de relajación (o en teoría, según el autor, este debiera ser su finalidad). Por esto es que el trabajo acompañado de música es un incentivo para un aumento de producción. El problema se sucede cuando en una compañía (en este caso, una compañía es tratada como una pequeña red de trabajo) todos los trabajadores escuchan su propia música, creándose, en vez de un tranquilo y placentero lugar de labores, en un inentendible bullicio.
¿Entonces, por que no centralizar la música?. El proyecto que se detalla en este informe mostrará la base o ideas generales en la que esta basado el software, llamado *329Player* (el nombre es en parte, un recordatorio del software en que esta basado, mpg123, y el nombre del ramo), el cual pretende dar solución a este problema planteado en forma inicial.

3. *Solución inicial*

3.1. *Pasos a seguir*

En la solución al problema, se plantea el siguiente esquema de trabajo:

- Elegir el tipo de interfaz. En teoría este debiera ser el último paso en la creación de un software, pero debido al desconocimiento inicial por parte del autor respecto a este tema, fue lo primero que se abordó.
- Estudiar el método para realizar la descompresión de sonido. Esto se explicará más detalladamente cuando se plantee la solución inicial.
- Realización de la parte de reproducción de sonido.
- Elección de protocolo de comunicación.
- Creación del software cliente y servidor.

3.2. *Problemas presentados y soluciones adoptadas.*

En el punto de selección de interfaz, se optó por una en consola, debido a que solo algunos equipos trabajan en modo gráfico (pensemos en empresas dedicadas al desarrollo, donde los usuarios trabajan en computadores que solo trabajan en este modo). Además, este tipo de interfaces pueden ser utilizadas en modos gráficos.

El tema de la descompresión fue el gran problema al momento de iniciar el trabajo. Esto, debido a que mp3 es un formato de audio que viene comprimido, el cual no es soportado directamente por las tarjetas de sonido, o solo por un número reducido de ellas (las cuales tienen un costo superior). Debido a esto, es que inicialmente, la solución era crear una librería, la cual debía tomar el archivo a reproducir y descomprimirlo en runtime. Esto obviamente es un trabajo grande, por lo cual esta solución fue rápidamente desechada. La pregunta era como lograr la descompresión sin tener que programar esta librería. La decisión final fue utilizar un programa, llamado *mpg123.* El por que de esta elección, se verá en la sección 4.

La solución inicial fue abandonada en este punto, aunque muchas de las ideas surgidas hasta aquí, fueron utilizadas en la solución final.

4. Solución y desarrollo

Hasta este punto solo se tiene definida el tipo de interfaz que se utilizará. Como se explicó anteriormente la descompresión de audio se realizará por medio de *mpg123*. La selección de este programa se explica a continuación.

4.1. Mpg123

- Mpg123 posee licencia GNU, por lo tanto es gratuito.
- Es un desarrollo de programadores para programadores.
- Posee opciones para funcionamiento back-end.
- Tiene soporte para controladores OSS, ALSA, ARTS y otros.
- Se encuentra bien documentado.

4.2. Lenguaje de programación

Luego de tener seleccionada el tipo de interfaz y el modo de reproducción, se debía seleccionar el lenguaje de programación que se utilizaría. Se optó por C++, debido a que es el paso natural a seguir luego de C, además de que permite trabajar en forma fácil el hecho de distribuir el problema en pequeños grupos de trabajo. De acuerdo a esto, el problema se separó en cuatro grupos:

- Desarrollo de la interfaz.
- Reproducción del audio.
- Creación del servidor.
- Creación del cliente.

Todas estas implementaciones son manejadas por un archivo *main.cpp*. Este se preocupa de mostrar y manejar, de acuerdo al modo de funcionamiento, las opciones necesarias para el uso del software.
El como se trabajo en cada uno de estos bloques, se especifica a continuación.

4.3. Desarrollo de la interfaz.

Primero se debe seleccionar de qué forma se creará esta interfaz. El modo lógico es una ventana que muestre los archivos de audio disponibles y opciones para su selección y reproducción. Existe una librería llamada *ncurses*, la cual permite la creación de ventanas en modo consola. Esta librería será la base para la creación de la interfaz.
El resultado final de este punto se encuentra en el archivo *winlib.cpp*, el cual implementa la clase *Win*. Los métodos disponibles para esta clase son los que se listan a continuación:

- virtual void Init();
  Este método inicializa las variables necesarias para el funcionamiento del manejo de las ventanas.
- virtual void FileWin(string);
  Muestra una ventana con la lista de archivos que existen en el sistema. Es fundamental para el modo local.
- virtual void FileWin(map<int,string>,string);
  En modo cliente, se crea una ventana que muestra un lista de archivos disponibles en el servidor.
- virtual int GetCh();
  Recupera y maneja una entrada por parte del usuario.
- virtual void Kill();
  Finaliza la ventana.
- virtual void MoveWin(int);
  Permite la navegación por los archivos disponibles.
- virtual void ChDir();
  En el modo local, cambia el directorio de trabajo.
- virtual void WriteInfo(map<int,int>);
  Escribe información respecto al archivo que se encuentra en reproducción actualmente.
- virtual void ShowVol(int);
  Muestra información respecto al volumen del dispositivo de audio.
- virtual string ToPlay();
  Selecciona el archivo a reproducir o encolar en modo local.
- virtual int ToSend();
  Selecciona el archivo a reproducir o encolar en modo cliente.

No existen ventanas en modo servidor, pues este modo no lo utiliza.

4.4. Reproducción del audio.

Esto se implementa en el archivo *audiolib.cpp*. Este archivo implementa dos clases, la primera, denominada *Audio* inicializa el servidor de sonido del sistema. El utilizado para este programa corresponde a arts. No existen mayores fundamentos para el uso de este, ya que también se podría haber utilizado algún otro, como esound. También se implementa la clase *Mp3*, la cual se preocupa de iniciar el programa utilizado en back-end, *mpg123*. Una lista de los métodos implementados por cada una de estas clases, es la siguiente:

Audio:

- virtual int SetVolume(int);
  Permite manejar el volumen del dispositivo de audio. Esto se realiza a través de la instrucción *ioctl()*.
- virtual int GetVolume();
  Obtiene información respecto al volumen del dispositivo.
- virtual bool Init();
  Inicializa la variables, además de iniciar el servidor de sonido. Esto se hace en un proceso hijo, tras una llamada a *fork()*.
- virtual void Kill();
  Finaliza el daemon de sonido, enviando una señal de término al proceso hijo.

Mp3:

- virtual bool Init();
  Inicialización de variables y lanzamiento de *mpg123* tras una llamada a *fork()*. La comunicación con este proceso se realizará por medio de un socket, el cual se crea por medio de *socketpair()*.
- virtual void Open(string);
  Abre y comienza a reproducir un archivo de audio.
- virtual void Action(string,int);
  Envia una señal a *mpg123*, las cuales pueden ser STOP, PAUSE, QUIT, JUMP. La señal STOP, detiene la repoducción en curso. PAUSE, pone en estado de pausa. QUIT finaliza el programa *mpg123* (no 329Player). JUMP avanza una cierta cantidad de frames, la que en este caso se fijo en +200 o -200. Esto corresponde a realizar un forward o un rewind en el archivo que se encuentra en reproducción. Además se realizan las acciones de NEXT y

PREV, para adelantar o retroceder en los temas que se encuentran actualmente en la cola de reproducción.

- virtual bool GetInfo();
  Se comunica con el proceso que mantiene ejecutando el software ya nombrado, y obtiene información del archivo que se esta reproduciendo actualmente.
- virtual void HandleInfo(map<int,int>&FrameInfo);
  Maneja la información obtenida, luego de un llamado a *GetInfo()*.
- virtual bool Add(string);
  Encola un archivo.
- virtual void Kill();
  Finaliza.

4.5. Aplicación Cliente-Servidor.

Esta es una parte que no se logró concluir, pero aquí se mostrará la idea de funcionamiento, para un futuro avance en esta área.
El funcionamiento del servidor, consiste en mantener una lista de archivos disponibles. Esta lista es entregada a los clientes a medida que estos la solicitan. La lista no necesariamente debe mostrar la estructura de directorios del servidor, pues esta puede ser mantenida en un archivo aparte, o encontrarse en una variable del programa. Claramente, por razones de escalabilidad, es preferible mantenerlos en un archivo.
El servidor maneja los requerimientos del cliente en forma similar a como son manejadas cuando el programa funciona en modo local. Es decir, si el cliente presiona la tecla X, que corresponde a PLAY en modo local, esta tecla es enviada al servidor, el cual la tratará de ejecutar esta opción, o colocarla en la cola de reproducción. Este funcionamiento, puede ser visto como el uso de un wurlitzer.
El cliente, por su parte, también simulará un funcionamiento en modo local, solo que el no realizará un manejo de solicitudes, pues estas serán enviadas al servidor.
Este funcionamiento se encuentra en fase de desarrollo en los archivos *clientlib.cpp* y *serverlib.cpp*. Los métodos, implementados por las clases Server y Client

Client:

- virtual bool Init(string,int);
  Inicialización de variables y conexión con servidor.
- virtual bool Request(int,int);
  Realización de requerimiento.

- virtual bool Answer(map<int,string>&Lista);
  Respuesta por parte del servidor.

Server:

- virtual bool Init(string,int);
  Inicializan del servidor.
- virtual string Accept();
  Aceptar solicitudes (dentro de un ciclo repetetivo). Esto idealmente debiera ser manejado a través de una señal *SIGIO*.
- virtual bool Response();
  Responder al cliente.
- virtual string ToPlay(string);
  Encolar o reproducir archivo de audio.

5. Continuación

El proyecto se presenta como una posibilidad de desarrollo a gran escala.
Para continuar el trabajo en esta área, se plantean los siguientes objetivos:

- Finalizar la comunicación con respecto a la aplicación en funcionamiento cliente-servidor.
- Realización de la librería de descompresión, para lograr un funcionamiento independiente.
- Mejorar el manejo de los archivos por parte del servidor. Esto, como planteamiento personal del autor, se puede realizar por medio del manejo de base de datos, como MySQL o PostgreSQL, los cuales también poseen licencia GNU.
- Trabajar en una interfaz X.

6. *Funcionamiento.*

La imagen muestra el software en pleno funcionamiento.

Figura 1: 329Player en modo local