ESCUELA DE INGENIERÍA EN COMPUTACIÓN

Sistemas Operativos

PROYECTO DE STREAMING

Profesor:
Esteban Arias

Equipo:
Esteban González Damazio 2014098850
Markus Villalobos Corrales 2013112890
Jose Pablo Vargas Campos 2013116365

I semestre 2018

Martes 5 de junio
San José, Costa Rica

Abstract

This project implements a server in the C language that permits streaming video files from a client. The client could be written in any language as long as it follows the protocol defined by the server. The streaming is done through the RTSP/RTP protocol, which stands for Real Time Streaming Protocol and Real Time Transport Protocol respectively. These two protocols in essence are in charge of the data stream communication over the web (RTP) and the control of said stream through messages that indicate information such as playing the stream from a certain point, restarting the stream or pausing it (RTSP). The implementation of the RTSP and RTP protocols is done with the help of the Gstreamer library, more information in the bibliography. On top of this implementation sits a communication protocol via sockets, it utilizes the library ZeroMQ to create a robust atomic and simplified control of sockets, the sockets communicate through the TCP protocol to message the server, this allows the client to communicate information about the username connecting, as well as ask for the available streams to which the server responds appropriately. Finally, the client, implemented in Java, creates a video sink to capture the stream of video data through the use of the JVLC library, this allows to playback the data packets.

Diseño

El sistema consta de dos partes, Servidor y Clientes. El servidor se desarrolla en C y vive dentro de una máquina remota host que corre el sistema operativo Debian y se encuentra en los servidores de DigitalOcean. El servidor es headless. Se aprovechan los protocolos RTSP/RTP (Real Time Streaming Protocol/Real-time Transport Protocol) para la comunicación de datos sobre la red. El protocolo RTSP permite la transmisión de datos para realizar streaming sin necesitar enviar todo el archivo. RTSP también se encarga de controlar el control de playback del stream entre los endpoints con instrucciones para pausar, reanudar o elegir un nuevo punto para el playback del archivo. Además se utilizan sockets para controlar la comunicación de datos que no son de streaming entre el Servidor y los Clientes, estos sockets se han utilizado con la ayuda de la librería opensource de ZeroMQ que tiene altos niveles de calidad y robusticidad. El cliente por otra parte se ha desarrollado en java, con su propia biblioteca en java de ZeroMQ para comunicarse con el servidor y con un sink de la

biblioteca de Jvlc para reproducir videos, la cual está basada en el reproductor open source VLC.

Bibliotecas

Gstreamer

https://gstreamer.freedesktop.org/documentation/index.html

Biblioteca en C que se utiliza para realizar las operaciones de servidor que se encargan de codificar los datos de video MP4, empaquetarlos a RTP y controlar las llamadas de control de playback RTSP. También se encarga de hacer lo mismo para los datos de audio Vorbis Ogg, sin embargo las limitaciones de la biblioteca cliente Jvlc no permiten utilizar los datos de audio Vorbis Ogg (.ogg).

Jvlc

https://github.com/caprica/vlcj

Biblioteca en Java que permite implementar VLC dentro de la vista de Swing. Se utiliza como el sink para poder desempacar los paquetes RTP, decodificar los datos de video en formato MP4 y controlar las llamadas RTSP. VLC tiene limitaciones para procesar los datos de audio Vorbis Ogg por lo que no puede utilizarse para la implementación de playback de música.

ZeroMQ

http://zeromq.org

ZeroMQ es una biblioteca de mensajería asíncrona de alto rendimiento, diseñada para su uso en aplicaciones distribuidas o concurrentes. Proporciona una cola de mensajes, pero a diferencia del middleware orientado a mensajes, el sistema ZeroMQ puede ejecutarse sin un intermediario de mensajes dedicado.

Servidor

Se compone de dos partes. La primera parte utiliza la biblioteca Gstreamer para realizar la comunicación de datos por TCP/IP utilizando el protocolo RTP/RTSP como una capa de abstracción sobre TCP/IP. El servidor es thread-safe y puede atender a múltiples

clientes al mismo tiempo sin problemas de concurrencia aun cuando diferentes clientes solicitan datos del mismo archivo. La segunda parte utiliza la biblioteca Zeromq para realizar el intercambio de mensajes con cada cliente por individual sin problema de confusión de clientes ni confusión de mensajes.

Localización del servidor

El servidor se ha alojado dentro de una máquina virtual corriendo Debian dentro de los servidores de la compañía DigitalOcean, esto nos permite darle al servidor un IP estático y asegurarnos de que siempre está disponible para los clientes. El IP del servidor es `206.189.228.117`.

Protocolo implementado

El protocolo se ha implementado por medio de mensajes entre el cliente y el servidor por medio de mensajes.

Si el mensaje del cliente indica "urls:" se está comunicando para preguntar por los URLs disponibles en el servidor (vídeos disponibles) seguido por un indice, indice 0 indica el primer video y así sigue incrementando hasta agotar las opciones.
Si pide un índice mayor a la cantidad de videos se comunica de vuelta con un "end:" indicandoles al cliente que no hay más videos disponibles.
El mensaje "name:" es similar al mensaje "urls:" pero en vez de comunicar URLs este mensaje comunica el nombre del video, el cual difiere al URL El cliente realiza un stream por medio del URL comunicado por el servidor.
Si pide un índice mayor a la cantidad de videos se comunica de vuelta con un "end:" indicandoles al cliente que no hay más videos disponibles.
El mensaje "user:" le indica al servidor el nombre del cliente conectado, el servidor devuelve un mensaje diciendo “Hello [nombre del user]”
El mensaje "play:" le indica al servidor el archivo de video del cual el cliente ha decidido realizar streaming.

Resumen de archivos

gstserver**urlgeneration.h**: utilizado para generar los URL de los streams, al tener su propio archivo asegura consistencia entre los varios programas que lo usan.

gstserver**zhelpers.h**: funciones útiles para ser reutilizadas constantemente junto a la biblioteca zeromq.

gstserver**mp4-streaming.c (-o mp4)**: La parte del servidor que hace el streaming. Programa que se encarga de crear los streams y asignar URLs. Utiliza `find` junto con pipes para crear los streams dinámicamente. Utiliza la biblioteca gstreamer para realizar el streaming del payload por medio del protocolo RTSP hacia el cliente por medio de un URL creado dinámicamente para cada archivo.

gstserver**unused.c**: Macros para evitar warning de gcc sobre parametros de funcion no utilizados.

gstserver**server.c (-o server)**: El servidor encargado de las conexiones de los clientes, realiza tareas como comunicar la lista de canciones, y en general la transferencia de mensajes entre el servidor y el usuario por medio de zeromq.

gstserver**makefile**: Makefile para construir el proyecto.

gstserver**run.sh**: Script de shell para correr el servidor el cual consiste en dos programas. Se crea un log dentro de un archivo de texto al redireccionar la salida estándar. Contiene los siguientes comandos:

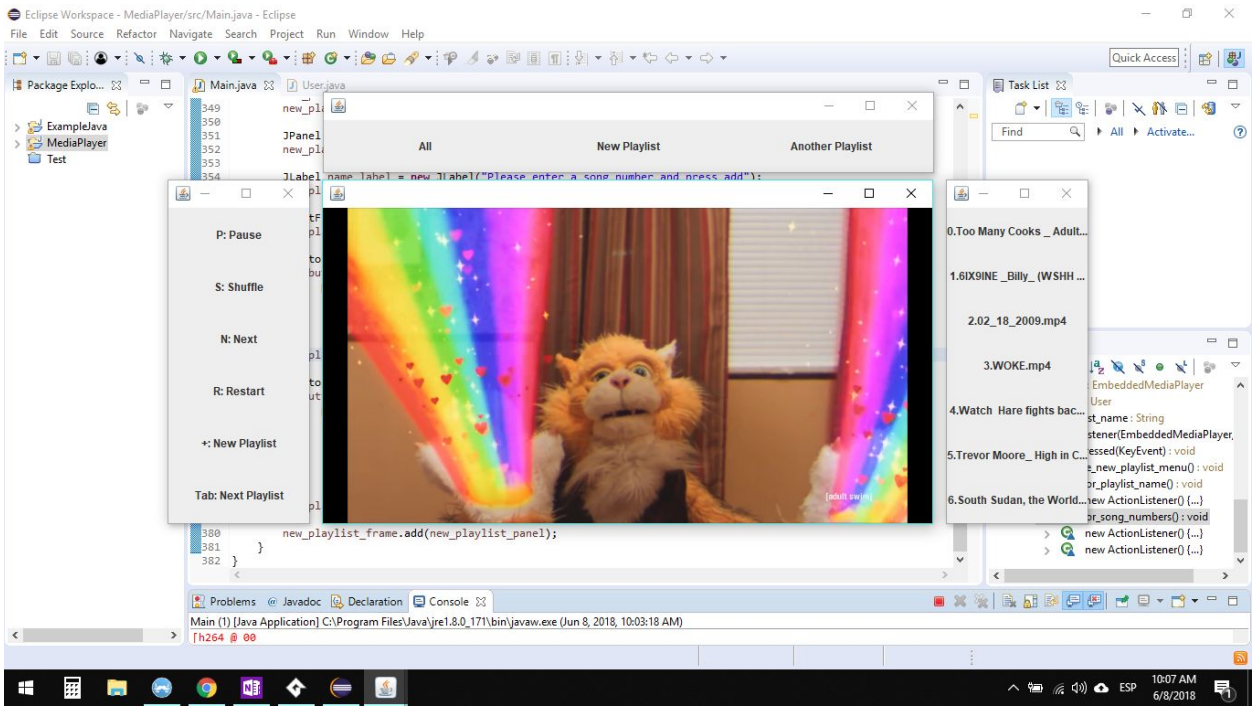
```
#!/bin/
bash

trap "trap - SIGTERM && kill -- -$$" SIGINT SIGTERM EXIT

find video -type f -print0 | xargs -0 ./mp4 > stream.log & ./server >
server.log & tail -f server.log
```

Cliente

MediaPlayer\src**Main.java** & MediaPlayer\src**User.java**

Desarrollado en Java con la biblioteca Jvlc. Se comunica con el servidor para recibir la lista de links para streaming de los datos, además puede enunciar su usuario y enunciar cuales vídeos quiere streamer para el log del servidor.

A continuación se muestra una imagen de como se ve el cliente activo:



En el panel izquierdo se ven los controles de la aplicación, el panel superior contiene los playlists creados por el usuario, y el panel derecho las canciones del playlist actual.

Pruebas Realizadas

Comunicación de mensajes: Se ha probado el servidor con 5 clientes concurrentes en un loop donde cada uno envía un mensaje diferente y como petición cada uno debe de recibir su respuesta apropiada del servidor. De esta manera se ha confirmado que el servidor puede manejar una demanda de mensajes concurrentes sin confundir la respuesta a ninguno de los mensajes. Entre los clientes se han utilizado clientes simplificados de C y de Java al mismo tiempo para realizar esta prueba.

gstserver**client**: cliente ejemplo en c utilizado para pedir mensajes

ExampleJava\src**main.java**: programa ejemplo en java para pedir mensajes

Visualización de stream RTSP:

Se ha probado la visualización de los streams RTSP desde el cliente de Java y desde el reproductor abierto VLC. Se ha probado la utilización de estos estos streams viendo el mismo archivo desde varios clientes diferentes para confirmar la habilidad de hacer streaming concurrente sin problemas entre varios clientes que requieren diferentes mensajes.

Estado del proyecto

Toda su funcionalidad está completa, el streaming se realiza con archivos de vídeo en vez de archivos de audio como extra.

Posibles mejoras

Recibir el stream de música por medio de esta biblioteca de RTSP/RTP para Java
http://www.smaxe.com/product.jsf?id=juv-rtsp-client
Junto a esta biblioteca que proporciona codecs para para Java del formato de audio Ogg Vorbis
http://www.jcraft.com/jorbis/
El servidor ya tiene la habilidad para originar el stream RTSP/RTP de audio necesario.

Bibliografia

Real Time Streaming Protocol (RTSP)
https://tools.ietf.org/html/rfc2326

Servidor de RTSP dentro de Gstreamer.
https://github.com/thaytan/gst-rtsp-server/tree/master/docs

Libreria Gstreamer para generar pipelines de archivos multimedia.
https://gstreamer.freedesktop.org/documentation/

Popen() para correr comandos y parsear su output. Crea un pipe, hace un fork e invoca al shell.
http://man7.org/linux/man-pages/man3/popen.3.html

VLCJ libreria para utilizar la habilidad de sink de VLC player dentro de una aplicación Java
https://github.com/caprica/vlcj

Anexo

run.sh

gstserver**run.sh**
Script encargado de correr el servidor, se corren dos programas ya que el servidor está dividido entre la parte que hace streaming de datos de video y el que comunica los datos con el cliente. Aquí se demuestra una implementación diferente entre ambos programas, el programa encargado de hacer el streaming de datos `./mp4` recibe los datos de los vídeos disponibles por medio de la línea de comandos, por lo tanto se ingresan por medio del comando `find video -type f -print0 | xargs -0` utilizando `find` para encontrar los archivos con la opción de terminar cada argumento con `null` en vez de `newline`, luego se utiliza `xargs` para realizar el pipe de los datos hacia el programa de manera correcta. A `./server` no hace falta enviar los datos ya que el los adquiere por medio de la función POSIX popen(), que tiene permite un funcionamiento similar a enviar los argumentos por medio de pipe pero se corre dentro del programa mismo. Se crea un log dentro de un archivo de texto al redireccionar la salida estandard. Finalmente `trap "trap - SIGTERM && kill -- -$$" SIGINT SIGTERM EXIT` nos permite capturar los procesos dentro del mismo proceso que el script, de esta manera al eliminar el script podemos eliminar los procesos de ambos programas del servidor.

```
#!/bin/bash

trap "trap - SIGTERM && kill -- -$$" SIGINT SIGTERM EXIT

find video -type f -print0 | xargs -0 ./mp4 > stream.log & ./server >
server.log & tail -f server.log
```

mp4-streaming.c

gstserver**mp4-streaming.c**

La parte del servidor encargada del streaming de datos de los videos mp4, gran parte del código de este archivo es para configurar de manera en que funciona la libreria gstserver y por lo tanto es código estándar utilizado en tal servidor, sin embargo fue modificado para responder a muchos archivos de video. Se encarga de crear los URLs para streaming y enviar los datos.

```
 /* GStreamer
 * Copyright (C) 2008 Wim Taymans <wim.taymans at gmail.com>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
```

```
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the
 * Free Software Foundation, Inc., 51 Franklin St, Fifth Floor,
 * Boston, MA 02110-1301, USA.
 */

#include "unused.h"

#include "urlgeneration.h"

#include <stdio.h>

#include <gst/gst.h>

#include <gst/rtsp-server/rtsp-server.h>

static char *port = (char *) DEFAULT_RTSP_PORT;

static GOptionEntry entries[] = {
  {"port", 'p', 0, G_OPTION_ARG_STRING, &port,
      "Port to listen on (default: " DEFAULT_RTSP_PORT ")", "PORT"},
  {NULL}
};

/* called when a stream has received an RTCP packet from the client */
static void
on_ssrc_active (GObject * session, GObject * source, GstRTSPMedia *
UNUSED(media))
{
  GstStructure *stats;

  GST_INFO ("source %p in session %p is active", (void*)source,
(void*)session);

  g_object_get (source, "stats", &stats, NULL);
```

```
  if (stats) {
    gchar *sstr;

    sstr = gst_structure_to_string (stats);
    g_print ("structure: %s\n", sstr);
    g_free (sstr);

    gst_structure_free (stats);
  }
}

static void
on_sender_ssrc_active (GObject * session, GObject * source, GstRTSPMedia *
UNUSED(media))
{
  GstStructure *stats;

  GST_INFO ("source %p in session %p is active", (void*)source,
(void*)session);

  g_object_get (source, "stats", &stats, NULL);
  if (stats) {
    gchar *sstr;

    sstr = gst_structure_to_string (stats);
    g_print ("Sender stats:\nstructure: %s\n", sstr);
    g_free (sstr);

    gst_structure_free (stats);
  }
}

/* signal callback when the media is prepared for streaming. We can get the
 * session manager for each of the streams and connect to some signals. */
static void
media_prepared_cb (GstRTSPMedia * media)
{
  guint i, n_streams;

  n_streams = gst_rtsp_media_n_streams (media);

  GST_INFO ("media %p is prepared and has %u streams", (void*)media,
n_streams);
```

```
  for (i = 0; i < n_streams; i++) {
    GstRTSPStream *stream;
    GObject *session;

    stream = gst_rtsp_media_get_stream (media, i);
    if (stream == NULL)
      continue;

    session = gst_rtsp_stream_get_rtpsession (stream);
    GST_INFO ("watching session %p on stream %u", (void*)session, i);

    g_signal_connect (session, "on-ssrc-active",
        (GCallback) on_ssrc_active, media);
    g_signal_connect (session, "on-sender-ssrc-active",
        (GCallback) on_sender_ssrc_active, media);
  }
}

static void
media_configure_cb (GstRTSPMediaFactory * factory, GstRTSPMedia * media)
{
  /* connect our prepared signal so that we can see when this media is
   * prepared for streaming */
  g_signal_connect (media, "prepared", (GCallback) media_prepared_cb,
factory);
}


int
main (int argc, char *argv[])
{
  GMainLoop *loop;
  GstRTSPServer *server;
  GstRTSPMountPoints *mounts;
  GstRTSPMediaFactory *factory;
  GOptionContext *optctx;
  GError *error = NULL;
  gchar *str;



  optctx = g_option_context_new ("<filename.mp4> - Test RTSP Server, MP4");
```

```
  g_option_context_add_main_entries (optctx, entries, NULL);
  g_option_context_add_group (optctx, gst_init_get_option_group ());
  if (!g_option_context_parse (optctx, &argc, &argv, &error)) {
    g_printerr ("Error parsing options: %s\n", error->message);
    g_option_context_free (optctx);
    g_clear_error (&error);
    return -1;
  }

  if (argc < 2) {
    g_print ("%s\n", g_option_context_get_help (optctx, TRUE, NULL));
    return 1;
  }
  g_option_context_free (optctx);

  loop = g_main_loop_new (NULL, FALSE);

  /* create a server instance */
  server = gst_rtsp_server_new ();
  g_object_set (server, "service", port, NULL);

  /* get the mount points for this server, every server has a default object
   * that be used to map uri mount points to media factories */
  mounts = gst_rtsp_server_get_mount_points (server);
```

Aquí se le indica al servidor como crear los streams de video a partir de los archivos de video, se utiliza un for() para crear un stream de datos por cada video enviado como argumento por medio de la línea de comandos.

```
  for (int arg  = 1; arg<argc; ++arg ) { // for each arg
```

La funcion `g_strdup_printf` se encarga de crear el stream:

`filesrc location=\"%s\"`	indica el archivo a crear un stream de datos
`qtdemux name=d`	separa los datos de audio y video del stream por medio de un `demux` y se les asigna el nombre `d` al stream de datos para referenciar en la próxima instrucción

`d. ! queue`	las líneas que incluyen esta instrucción indica que se debe de realizar en paralelo, esto incluye `rtph264pay` y `rtpmp4apay` el primero representa los datos de sonido y el segundo los de video
`rtph264pay pt=96 name=pay0`	convierte los datos de sonido del archivo en un stream de datos crudo y le asigna el identificador de pay0
`rtpmp4apay pt=97 name=pay1`	convierte los datos de vídeo del archivo en un stream de datos crudo y le asigna el identificador de pay1
`argv[arg]`	se utiliza para indicar el archivo a la instruccion `filesrc location=\"%s\"`

```
    printf("%s\n",argv[arg]);
    str = g_strdup_printf ("( "
        "filesrc location=\"%s\" ! qtdemux name=d "
        "d. ! queue ! rtph264pay pt=96 name=pay0 "
        "d. ! queue ! rtpmp4apay pt=97 name=pay1 " ")", argv[arg]);

    /* make a media factory for a test stream. The default media factory can
use
     * gst-launch syntax to create pipelines.
     * any launch line works as long as it contains elements named pay%d.
Each
     * element with pay%d names will be a stream */
    factory = gst_rtsp_media_factory_new ();
    gst_rtsp_media_factory_set_launch (factory, str);
    g_signal_connect (factory, "media-configure", (GCallback)
media_configure_cb,
        factory);
    g_free (str);

    char url[URL_SIZE];
```

Aquí se utiliza la función `RemoveSpaces(argv[arg]);` que proviene de **urlgeneration.h** para generar URLs a partir de nombres de archivo. Luego la isntrsuccion `snprintf(url, sizeof(url), "/%s", argv[arg]);` genera el URL a una variable "url" y se crea el URL por medio de la instruccion `gst_rtsp_mount_points_add_factory (mounts, url, factory);` en este momento se está listo el stream de datos por medio del URL.

```
    RemoveSpaces(argv[arg]);
    snprintf(url, sizeof(url), "/%s", argv[arg]);
    g_print ("stream will be ready at rtsp://127.0.0.1:%s%s\n", port, url);

    /* attach the test factory to the url */
    gst_rtsp_mount_points_add_factory (mounts, url, factory);
  }

  /* don't need the ref to the mapper anymore */
  g_object_unref (mounts);

  /* attach the server to the default maincontext */
  gst_rtsp_server_attach (server, NULL);

  /* start serving */
  g_print ("streams ready!\n");
  g_main_loop_run (loop);

  return 0;
}
```

Server.c

gstserver**server.c**

Servidor encargado de comunicar con el cliente los urls para streaming y el nombre del cliente

```
#include <zmq.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include "zhelpers.h"
#include "urlgeneration.h"
#define REMOTE_IP "206.189.228.117"
#define LOCALHOST "127.0.0.1"

struct comType *comm, *comm1, *comm2;
char **urls;
int count =0;
```

Función para agregar un string al array de strings `char **urls;` aquí se guardan los nombres de los archivos de vídeo para comunicar con el cliente el nombre y el URL.

```
void add_string(char** array, int* size, const char* string)
{
   urls = realloc(array, (*size + 1) *sizeof(char*) );
   urls[*size] = malloc(strlen(string)+1);
   strcpy(urls[*size], string);
   *size += 1;
}
```

Función obtener los nombres de archivos dentro de la carpeta de videos del proyecto, se utiliza el comando `popen("find video -type f", "r");` para correr un `find` (comando de unix) y encontrar todos los videos dentro de la carpeta, esto nos permite crear un pipe que se comunica con este mismo programa.

```
void getUrls(){
  FILE *fp;
  int status;
  char path[URL_SIZE];
  fp = popen("find video -type f", "r");
  if (fp == NULL)
      /* Handle error */
      printf("error with fp");
  while (fgets(path, URL_SIZE, fp) != NULL){
      //printf("%s", path);
      add_string(urls,&count,path);
      //printf("1 %d\n", count);
      //printf("str %s\n",urls[count-1]);
  }
  status = pclose(fp);
  if (status == -1) {
      /* Error reported by pclose() */
      printf("error with pclose()");
  } else {
      /* Use macros described under wait() to inspect `status' in order
         to determine success/failure of command executed by popen() */
  }
}
```

Main loop del server, aqui se realiza la comunicación.

```
int main (void)
{
    printf("Sessions server.\n");
    fflush(stdout);
```

Se crea el socket para comunicarse con los clientes según lo indica la biblioteca de zeromq.

```
    //  Socket to talk to clients
    void *context = zmq_ctx_new ();
    void *responder = zmq_socket (context, ZMQ_REP);
    int rc = zmq_bind (responder, "tcp://*:5555");
    assert (rc == 0);

    getUrls();
```

Cada vez que entra al loop indica la conexión de 1 cliente. La función `s_recv` se encarga de recibir el mensaje, la función está implementada en `zhelpers.h` para acortar el código dentro del código principal, permite verificar el formato del mensaje.

```
    while (1) {
        char *string = s_recv (responder);
        char url[URL_SIZE];
        strcpy(url, "world");
        printf ("Received: %s.\n", string);     //  Show progress
```

Si el mensaje del cliente indica `"urls:"` se esta comunicando para preguntar por los URLs disponibles en el servidor (vídeos disponibles) seguido por un indice, indice 0 indica el primer video y así hasta agotar las opciones.

```
        if(strstr(string, "urls:") != NULL){
          if(string[5]-'0'<count){
            sprintf(url,"rtsp://%s:8554/%s", REMOTE_IP, urls[string[5]-'0']);
            RemoveSpaces(url);
          }
          else{
```

Si pide un índice mayor a la cantidad de videos se comunica de vuelta con un `"end:"` indicandoles al cliente que no hay más videos disponibles.

```
            strcpy(url,"end:");
          }
        }
```

El mensaje `"name:"` es similar al mensaje `"urls:"` pero en vez de comunicar URLs este mensaje comunica el nombre del video, el cual difiere al URL

```
        if(strstr(string, "name:") != NULL){
          if(string[5]-'0'<count){
            sprintf(url,"%s", urls[string[5]-'0']+6);
          }
```

Si pide un índice mayor a la cantidad de videos se comunica de vuelta con un `"end:"` indicandoles al cliente que no hay más videos disponibles.

```
          else{
            strcpy(url,"end:");
          }
        }
```

El mensaje `"user:"` le indica al servidor el nombre del cliente conectado, el servidor devuelve un mensaje diciendo “Hello [nombre del user]”

```
        if(strstr(string, "user:") != NULL){
            sprintf(url, "Hello, %s!", string+5);
        }
```

El mensaje `"play:"` le indica al servidor el archivo de video del cual el cliente ha decidido realizar streaming.

```
        if(strstr(string, "play:") != NULL){
            sprintf(url, "You are now playing: %s!", string+5);
        }
        fflush(stdout);
        free (string);
        //sleep (1);
```

La función `s_send` se encarga de enviar el mensaje, la función está implementada en `zhelpers.h` para acortar el código dentro del código principal, permite verificar el formato del mensaje.

```
        s_send (responder, url);        //  Send results to sink
    }
    zmq_close (responder);
    zmq_ctx_destroy (context);
    return 0;
}
```

Client.c

gstserver**client.c**

Cliente de pruebas escrito en C, se corrieron múltiples instancias de este cliente al mismo tiempo junto al cliente en java para asegurar que los mensajes de los clientes no fueran confundidos entre sí. Manda un mensaje simple que espera de regreso siempre el mismo mensaje, sin embargo el cliente en java esperaba mensajes diferentes, si se confunde al cliente a quien enviar mensajes entonces el cliente en java recibiría mensajes de “World”, lo cual sucedió al realizar la implementación de manera incorrecta pero luego fue corregido.

```
//  Hello World client
//  Connects REQ socket to tcp://localhost:5559
//  Sends "Hello" to server, expects "World" back

#include "zhelpers.h"

int main (void)
{
    void *context = zmq_ctx_new ();

    //  Socket to talk to server
    void *requester = zmq_socket (context, ZMQ_REQ);
    zmq_connect (requester, "tcp://localhost:5555");

    int request_nbr;
    for (request_nbr = 0; request_nbr != 10; request_nbr++) {
        s_send (requester, "Hello");
        char *string = s_recv (requester);
        printf ("Received reply %d [%s]\n", request_nbr, string);
        free (string);
    }
    zmq_close (requester);
    zmq_ctx_destroy (context);
    return 0;
}
```

zhelpers.c

gstserver**zhelpers.c**

Archivo utilizado para asegurar la integridad de mensajes de zeromq, se utiliza para reducir el código el archivo del servidor

```
/*  =====================================================================
    zhelpers.h
    Helper header file for example applications.
    =====================================================================
*/

#ifndef __ZHELPERS_H_INCLUDED__
#define __ZHELPERS_H_INCLUDED__

//  Include a bunch of headers that we will need in the examples

#include <zmq.h>

#include <assert.h>
#include <signal.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#if (!defined (WIN32))
#   include <sys/time.h>
#endif

#if (defined (WIN32))
#   include <windows.h>
#endif

//  Version checking, and patch up missing constants to match 2.1
#if ZMQ_VERSION_MAJOR == 2
#   error "Please upgrade to ZeroMQ/3.2 for these examples"
#endif

//  On some version of Windows, POSIX subsystem is not installed by default.
//  So define srandom and random ourself.
```

```
#if (defined (WIN32))
#   define srandom srand
#   define random rand
#endif

//  Provide random number from 0..(num-1)
#define randof(num)  (int) ((float) (num) * random () / (RAND_MAX + 1.0))

//  Receive 0MQ string from socket and convert into C string
//  Caller must free returned string. Returns NULL if the context
//  is being terminated.
static char *
```

La función s_recv es una de las funciones más utilizadas de este archivo, se encarga de recibir mensajes con el formato correcto.

```
s_recv (void *socket) {
    char buffer [256];
    int size = zmq_recv (socket, buffer, 255, 0);
    if (size == -1)
        return NULL;
    buffer[size] = '\0';

#if (defined (WIN32))
    return strdup (buffer);
#else
    return strndup (buffer, sizeof(buffer) - 1);
#endif

    // remember that the strdup family of functions use malloc/alloc for space for the new string.  It
must be manually
    // freed when you are done with it.  Failure to do so will allow a heap attack.
}

//  Convert C string to 0MQ string and send to socket
```

La función s_send es la otra funcion más utilizada de este archivo, se encarga de mandar mensajes con el formato correcto.

```
static int
s_send (void *socket, char *string) {
    int size = zmq_send (socket, string, strlen (string), 0);
```

```
    return size;
}

//  Sends string as 0MQ string, as multipart non-terminal
static int
s_sendmore (void *socket, char *string) {
    int size = zmq_send (socket, string, strlen (string), ZMQ_SNDMORE);
    return size;
}

//  Receives all message parts from socket, prints neatly
//
static void
s_dump (void *socket)
{
    int rc;

    zmq_msg_t message;
    rc = zmq_msg_init (&message);
    assert (rc == 0);

    puts ("----------------------------------------");
    //  Process all parts of the message
    do {
        int size = zmq_msg_recv (&message, socket, 0);
        assert (size >= 0);

        //  Dump the message as text or binary
        char *data = (char*)zmq_msg_data (&message);
        assert (data != 0);
        int is_text = 1;
        int char_nbr;
        for (char_nbr = 0; char_nbr < size; char_nbr++) {
            if ((unsigned char) data [char_nbr] < 32
                || (unsigned char) data [char_nbr] > 126) {
                is_text = 0;
            }
        }

        printf ("[%03d] ", size);
        for (char_nbr = 0; char_nbr < size; char_nbr++) {
            if (is_text) {
                printf ("%c", data [char_nbr]);
```

```
            } else {
                printf ("%02X", (unsigned char) data [char_nbr]);
            }
        }
        printf ("\n");
    } while (zmq_msg_more (&message));

    rc = zmq_msg_close (&message);
    assert (rc == 0);
}

#if (!defined (WIN32))
//  Set simple random printable identity on socket
//  Caution:
//    DO NOT call this version of s_set_id from multiple threads on MS Windows
//    since s_set_id will call rand() on MS Windows. rand(), however, is not
//    reentrant or thread-safe. See issue #521.
static void
s_set_id (void *socket)
{
    char identity [10];
    sprintf (identity, "%04X-%04X", randof (0x10000), randof (0x10000));
    zmq_setsockopt (socket, ZMQ_IDENTITY, identity, strlen (identity));
}
#else
//  Fix #521 for MS Windows.
static void
s_set_id(void *socket, intptr_t id)
{
    char identity [10];
    sprintf(identity, "%04X", (int)id);
    zmq_setsockopt(socket, ZMQ_IDENTITY, identity, strlen(identity));
}
#endif

//  Sleep for a number of milliseconds
static void
s_sleep (int msecs)
{
#if (defined (WIN32))
    Sleep (msecs);
#else
    struct timespec t;
```

```
    t.tv_sec  =  msecs / 1000;
    t.tv_nsec = (msecs % 1000) * 1000000;
    nanosleep (&t, NULL);
#endif
}

//  Return current system clock as milliseconds
static int64_t
s_clock (void)
{
#if (defined (WIN32))
    SYSTEMTIME st;
    GetSystemTime (&st);
    return (int64_t) st.wSecond * 1000 + st.wMilliseconds;
#else
    struct timeval tv;
    gettimeofday (&tv, NULL);
    return (int64_t) (tv.tv_sec * 1000 + tv.tv_usec / 1000);
#endif
}

//  Print formatted string to stdout, prefixed by date/time and
//  terminated with a newline.

static void
s_console (const char *format, ...)
{
    time_t curtime = time (NULL);
    struct tm *loctime = localtime (&curtime);
    char *formatted = (char*)malloc (20);
    strftime (formatted, 20, "%y-%m-%d %H:%M:%S ", loctime);
    printf ("%s", formatted);
    free (formatted);

    va_list argptr;
    va_start (argptr, format);
    vprintf (format, argptr);
    va_end (argptr);
    printf ("\n");
}

#endif  //  __ZHELPERS_H_INCLUDED__
```

urlgeneration.c

gstserver**urlgeneration.c**

Archivo utilizado para la generación de URLs, aquí se definen cosas como el puerto y además una funciona para eliminar los espacios de un nombre de archivo, esto se comparte entre los dos programas encargados de funcionar como servidor para tener una implementación estándar de los datos para los URLs de streaming y asegurarse de que ambos programas servidor produzcan los mismos URLs.

```
#ifndef REMOVESPACES_H
#define REMOVESPACES_H

#define IP_EXAMPLE "rtsp://127.0.0.1:"
#define DEFAULT_RTSP_PORT "8554"
#define URL_SIZE 1024


void RemoveSpaces(char* source)
{
  char* i = source;
  char* j = source;
  while(*j != 0)
  {
    *i = *j++;
    if(*i != ' ')
      i++;
  }
  *i = 0;
}


#endif /* REMOVESPACES_H */
```

Main.java (cliente ejemplo)

ExampleJava\src**Main.java**

Cliente ejemplo en java muy simplificado, se utiliza para demostrar la comunicación con el server por medio de la consola, no tiene interfaz gráfica y su objetivo no es mostrar los videos, solamente la comunicación con el server, funciona para entender el ciclo de enviar y recibir mensajes. Su objetivo es ilustrativo y de enseñanza.

```
//
//  Hello World client in Java
//  Connects REQ socket to tcp://localhost:5555
//  Sends "Hello" to server, expects "World" back
//

import java.util.Objects;

import org.zeromq.ZMQ;
public class main {
        static String remote = "tcp://206.189.228.117:5555";
        static String local = "tcp://127.0.0.1:5555";

    public static void main(String[] args) {
        ZMQ.Context context = ZMQ.context(1);

        //  Socket to talk to server
        System.out.println("Connecting to hello world server…");

        ZMQ.Socket requester = context.socket(ZMQ.REQ);
        requester.connect(remote);

     // tcp://127.0.0.1:5555
     // tcp://206.189.228.117:5555
     // rtsp://206.189.228.117:8554/ [url]

        //este es un bloque ejemplo que envia "hello" y recibe "world"
        //importante: cada vez que hace un REQUEST >>tiene<< que hacer un RECEIVE
        // RECEIVE: "requester.recv(0);"
        for (int requestNbr = 0; requestNbr != 1; requestNbr++) {
            String request = "Hello";
            System.out.println("Sending: Hello " + requestNbr);
            requester.send(request.getBytes(), 0);

            byte[] reply = requester.recv(0);
            System.out.println("Received: " + new String(reply) + " " + requestNbr);
        }
```

```
// este es el bloque de codigo para pedir urls
byte[] reply = null;
int count=0;
//while:
// reply = lo que responde el servidor
// null = para la primera vez que se corre y aun no hay nada en el reply
// !Objects.equals("end:",new String(reply)) = terminar cuando el servidor responde
//                        con "end:"
// count = para indicarle al servidor por cual url vamos pidiendo

// el protocolo es el siguiente:
// enviar "urls:#" donde "#" es el indice del url por el que va.
// se devuelve el final de cada url, ese se le agrega al rtsp://ip:port/ + url
// cuando envia un "#" mayor a los urls disponibles el server devuelve la
// llamada "end:"
while(reply== null || !Objects.equals("end:",new String(reply))) {
        String request = "urls:"+count;
        System.out.println("Sending: urls:"+count);
        requester.send(request.getBytes(), 0);
        count++;
        reply = requester.recv(0);
        System.out.println("Received: " + new String(reply));
}

//se reinician estas cosas para poder hacer loop otra vez
reply = null;
count=0;

//igual que antes pero esta vez con nombres
while(reply== null || !Objects.equals("end:",new String(reply))) {
        String request = "name:"+count;
        System.out.println("Sending: name:"+count);
        requester.send(request.getBytes(), 0);
        count++;
        reply = requester.recv(0);
        System.out.println("Received: " + new String(reply));
}

//protocolo "user:<nombre>", sirve para que el server imprima en pantalla (terminal)
//que un user se ha conectado. Correr esto 1 vez al principio solo para cumplir
//con la especificacion del profe de enunciar los usuarios
String request = "user:Markus";
```

```
        System.out.println("Sending: "+request);
        requester.send(request.getBytes(), 0);
        count++;
        reply = requester.recv(0);
        System.out.println("Received: " + new String(reply));

        //protocolo "play:<nombre>", sirve para que el server imprima en pantalla (terminal)
        //cual cancion . Correr esto 1 vez con el url de el video a hacer play, solo para
        //cumplir con la especificacion de enunciar cuales videos se les hace play
        request = "play:Papa";
        System.out.println("Sending: "+request);
        requester.send(request.getBytes(), 0);
        count++;
        reply = requester.recv(0);
        System.out.println("Received: " + new String(reply));

        requester.close();
        context.term();
    }
}
```

MainJava.java (cliente real)

MediaPlayer\src**MainJava.java**

Código del Cliente

La labor del cliente es crear una interfaz gráfica para que el usuario pueda comunicarse con los datos que desea obtener del servidor. Además utiliza la misma funcionalidad que el cliente de ejemplo `ExampleJava` para comunicarse con el servidor.

Main:

```
import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

import javax.print.attribute.standard.MediaSize.Other;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

import com.sun.jna.Native;
import com.sun.jna.NativeLibrary;

import uk.co.caprica.vlcj.binding.LibVlc;
import uk.co.caprica.vlcj.player.MediaPlayer;
import uk.co.caprica.vlcj.player.MediaPlayerEventAdapter;
import uk.co.caprica.vlcj.player.MediaPlayerFactory;
import uk.co.caprica.vlcj.player.embedded.EmbeddedMediaPlayer;
import uk.co.caprica.vlcj.player.embedded.windows.Win32FullScreenStrategy;
import uk.co.caprica.vlcj.runtime.RuntimeUtil;

import java.util.ArrayList;
```

```
import java.util.Objects;

import org.zeromq.ZMQ;

import java.util.Scanner;


public class Main {

	static EmbeddedMediaPlayer emp;
	static User user;

	static int screenWidth;
	static int screenHeight;

	static JFrame video_frame;
	static JFrame songs_frame;
	static JFrame controls_frame;
	static JFrame playlists_frame;

	static ZMQ.Socket requester;

	static String ip;
	static String port;
	static String video_port;

	public static void main(String[] args){
		Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
		screenWidth = screenSize.width;
		screenHeight = screenSize.height;

		ip = "206.189.228.117";
		port = "5555";
		video_port = "8554";

		create_socket();
		create_user();
		create_playlists_frame();
		create_songs_frame();
		create_controls_frame();
		create_video_frame();
	}
```

```
    public static void create_socket() {
          ZMQ.Context context = ZMQ.context(1);
      requester = context.socket(ZMQ.REQ);
      requester.connect("tcp://"+ip+":"+port);
    }

    public static void create_user() {
          Scanner reader = new Scanner(System.in);
          System.out.print("Please enter your username: ");
          String name = reader.nextLine();
          reader.close();

          user = new User(name);

          byte[] reply = null;
      String request = "user:"+name;
      requester.send(request.getBytes(), 0);
      reply = requester.recv(0);
      System.out.println(new String(reply));

      ArrayList<Integer> all_playlist = new ArrayList<Integer>();
      reply = null;
      int count=0;
      while(reply == null || !Objects.equals("end:",new String(reply))) {
            request = "urls:"+count;
            requester.send(request.getBytes(), 0);
            reply = requester.recv(0);
            if(!Objects.equals("end:",new String(reply))) {
                all_playlist.add(count);
            }
            count++;
      }

      user.add_playlist("All",all_playlist,requester);

    }

  public static void create_playlists_frame() {
      playlists_frame = new JFrame();

playlists_frame.setLocation((screenWidth/2)-(screenWidth/4),(screenHeight/2)-
(screenHeight/4)-(screenHeight/8));
```

```
        playlists_frame.setSize((screenWidth/2),(screenHeight/8));
        playlists_frame.setVisible(true);

        JPanel playlists_panel = new JPanel(new
GridLayout(1,user.playlist_names_list.size()));

playlists_panel.setLocation(playlists_frame.getX(),playlists_frame.getY());

        for(int i=0;i<user.playlist_names_list.size();i++) {
            String temp_playlist_name = user.playlist_names_list.get(i);
            JLabel new_label = new JLabel(temp_playlist_name,JLabel.CENTER);
            playlists_panel.add(new_label);
        }

        playlists_frame.add(playlists_panel);
    }

      public static void create_songs_frame() {
            songs_frame = new JFrame();

songs_frame.setLocation((screenWidth/2)+(screenWidth/4),(screenHeight/2)-(scr
eenHeight/4));
            songs_frame.setSize((screenWidth/8),(screenHeight/2));
            songs_frame.setVisible(true);

            JPanel songs_panel = new JPanel(new
GridLayout(user.rtsp_list.size(),1));
            songs_panel.setLocation(songs_frame.getX(),songs_frame.getY());

        byte[] reply = null;
        for(int i=0;i<user.current_playlist.size();i++){
            int temp_song_number = user.current_playlist.get(i);
              String request = "name:"+temp_song_number;
              requester.send(request.getBytes(), 0);
              reply = requester.recv(0);
              if(!Objects.equals("end:",new String(reply))) {
                        JLabel new_label = new JLabel(i+"."+new
String(reply),JLabel.CENTER);
                        songs_panel.add(new_label);
              }
        }

            songs_frame.add(songs_panel);
```

```
	}

	public static void create_controls_frame() {
		controls_frame = new JFrame();

controls_frame.setLocation((screenWidth/2)-(screenWidth/4)-(screenWidth/8),(s
creenHeight/2)-(screenHeight/4));
		controls_frame.setSize((screenWidth/8),(screenHeight/2));
		controls_frame.setVisible(true);

		JPanel controls_panel = new JPanel(new GridLayout(6,1));

controls_panel.setLocation(controls_frame.getX(),controls_frame.getY());

		JLabel pause_label = new JLabel("P: Pause",JLabel.CENTER);
		controls_panel.add(pause_label);
		JLabel shuffle_label = new JLabel("S: Shuffle",JLabel.CENTER);
		controls_panel.add(shuffle_label);
		JLabel next_label = new JLabel("N: Next",JLabel.CENTER);
		controls_panel.add(next_label);
		JLabel restart_label = new JLabel("R: Restart",JLabel.CENTER);
		controls_panel.add(restart_label);
		JLabel new_playlist_label = new JLabel("+: New
Playlist",JLabel.CENTER);
		controls_panel.add(new_playlist_label);
		JLabel next_playlist_label = new JLabel("Tab: Next
Playlist",JLabel.CENTER);
		controls_panel.add(next_playlist_label);

		controls_frame.add(controls_panel);
	}

	public static void create_video_frame(){
		///GUI
		//Create Window
		video_frame = new JFrame();
		video_frame.setLocation((screenWidth/4),(screenHeight/4));
		video_frame.setSize(screenWidth/2,screenHeight/2);
		video_frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		video_frame.setVisible(true);

		//Create Canvas
		Canvas c = new Canvas();
```

```
            c.setBackground(Color.black);
            JPanel p = new JPanel();
            p.setLayout(new BorderLayout());
            p.add(c);
            video_frame.add(p);

            ///VLCJ
            //Load Library

NativeLibrary.addSearchPath(RuntimeUtil.getLibVlcLibraryName(),"lib");

Native.loadLibrary(RuntimeUtil.getLibVlcLibraryName(),LibVlc.class);
            //Initialize Media Player
            MediaPlayerFactory mpf = new MediaPlayerFactory();
            //Control User Interactions
            emp = mpf.newEmbeddedMediaPlayer(new
Win32FullScreenStrategy(video_frame));
            emp.setVideoSurface(mpf.newVideoSurface(c));
            //Full Screen
            //emp.toggleFullScreen();
            //Hide Cursor
            emp.setEnableMouseInputHandling(false);
            //Disable Keyboard
            emp.setEnableKeyInputHandling(false);

            String file = user.current_rtsp;
            //Prepare File
            emp.prepareMedia(file);
            //Read the File
            emp.play();

            //Add Key Listener
            KeyListener key_listener = new KeyListener(emp,user);
            video_frame.addKeyListener(key_listener);
            video_frame.setFocusTraversalKeysEnabled(false);
            c.addKeyListener(key_listener);
            c.setFocusTraversalKeysEnabled(false);
            p.addKeyListener(key_listener);
            p.setFocusTraversalKeysEnabled(false);

            emp.addMediaPlayerEventListener(new MediaPlayerEventAdapter() {
                @Override
                public void finished(MediaPlayer mediaPlayer) {
```

```
                    user.go_to_next_rtsp(emp);
                }
            });
        }
}

class KeyListener extends KeyAdapter {

        EmbeddedMediaPlayer emp;
        User user;
        String playlist_name;

        public KeyListener(EmbeddedMediaPlayer emp, User user) {
                this.emp = emp;
                this.user = user;
        }

        public void keyPressed(KeyEvent evt) {
                if (evt.getKeyChar() == 'r' || evt.getKeyChar() == 'R'){
                        emp.stop();
                        emp.play();
                }
                if (evt.getKeyChar() == 's' || evt.getKeyChar() == 'S'){
                        user.shuffle = !user.shuffle;
                }
                if (evt.getKeyChar() == 'n' || evt.getKeyChar() == 'N') {
                        user.go_to_next_rtsp(emp);
                }
                if (evt.getKeyChar() == 'p' || evt.getKeyChar() == 'P') {
                        if(emp.isPlaying()) {
                                emp.pause();
                        }else {
                                emp.play();
                        }
                }

                if (evt.getKeyChar() == '0'){
                        user.go_to_index_rtsp(emp, 0);
                }
                if (evt.getKeyChar() == '1'){
                        user.go_to_index_rtsp(emp, 1);
                }
                if (evt.getKeyChar() == '2'){
```

```
            user.go_to_index_rtsp(emp, 2);
        }
        if (evt.getKeyChar() == '3'){
            user.go_to_index_rtsp(emp, 3);
        }
        if (evt.getKeyChar() == '4'){
            user.go_to_index_rtsp(emp, 4);
        }
        if (evt.getKeyChar() == '5'){
            user.go_to_index_rtsp(emp, 5);
        }
        if (evt.getKeyChar() == '6'){
            user.go_to_index_rtsp(emp, 6);
        }
        if (evt.getKeyChar() == '7'){
            user.go_to_index_rtsp(emp, 7);
        }
        if (evt.getKeyChar() == '8'){
            user.go_to_index_rtsp(emp, 8);
        }
        if (evt.getKeyChar() == '9'){
            user.go_to_index_rtsp(emp, 9);
        }

    if (evt.getKeyChar() == '+') {
        emp.stop();
        user.change_playlist(0, Main.requester);
        Main.songs_frame.dispose();
        Main.create_songs_frame();
        create_new_playlist_menu();
    }

    if (evt.getKeyCode() == KeyEvent.VK_TAB) {
        user.go_to_next_playlist(Main.requester);
        Main.songs_frame.dispose();
        Main.create_songs_frame();
    }
}

private void create_new_playlist_menu() {
        ask_for_playlist_name();
}
```

```
      public void ask_for_playlist_name() {
        JFrame new_playlist_frame = new JFrame();

new_playlist_frame.setLocation((Main.screenWidth/2)-((Main.screenWidth/8)),(M
ain.screenHeight/2)-(Main.screenHeight/8));

new_playlist_frame.setSize((Main.screenWidth/4),(Main.screenHeight/4));
        new_playlist_frame.setVisible(true);

        JPanel new_playlist_panel = new JPanel(new GridLayout(3,1));

new_playlist_panel.setLocation(new_playlist_panel.getX(),new_playlist_panel.g
etY());

        JLabel name_label = new JLabel("Please enter the name of the new
playlist");
        new_playlist_panel.add(name_label);

        JTextField name_textfield = new JTextField();
        new_playlist_panel.add(name_textfield);

        JButton name_button = new JButton("Ok");
        name_button.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    playlist_name = name_textfield.getText();
                    ask_for_song_numbers();
                    new_playlist_frame.dispose();
              }
              });
        new_playlist_panel.add(name_button);

        new_playlist_frame.add(new_playlist_panel);
      }

      public void ask_for_song_numbers() {
            ArrayList<Integer> new_playlist = new ArrayList<Integer>();

        JFrame new_playlist_frame = new JFrame();

new_playlist_frame.setLocation((Main.screenWidth/2)-((Main.screenWidth/8)),(M
ain.screenHeight/2)-(Main.screenHeight/8));

new_playlist_frame.setSize((Main.screenWidth/4),(Main.screenHeight/4));
```

```
        new_playlist_frame.setVisible(true);

        JPanel new_playlist_panel = new JPanel(new GridLayout(5,1));

new_playlist_panel.setLocation(new_playlist_panel.getX(),new_playlist_panel.g
etY());

        JLabel name_label = new JLabel("Please enter a song number and press
add");
        new_playlist_panel.add(name_label);

        JTextField name_textfield = new JTextField();
        new_playlist_panel.add(name_textfield);

        JButton add_button = new JButton("Add");
        add_button.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {

new_playlist.add(Integer.parseInt(name_textfield.getText()));
                    name_textfield.setText("");
              }
              });
        new_playlist_panel.add(add_button);

        JButton ok_button = new JButton("Ok");
        ok_button.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    Main.user.add_playlist(playlist_name, new_playlist,
Main.requester);
                    Main.playlists_frame.dispose();
                    Main.create_playlists_frame();
                    new_playlist_frame.dispose();
              }
              });
        new_playlist_panel.add(ok_button);

        new_playlist_frame.add(new_playlist_panel);
      }
}
```

User:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.ThreadLocalRandom;

import org.zeromq.ZMQ;

import uk.co.caprica.vlcj.player.embedded.EmbeddedMediaPlayer;

public class User {

      public String name;

    public List<String> playlist_names_list;
    public String current_playlist_name;
    public List<List<Integer>> playlist_list;
    public List<Integer> current_playlist;
      public List<String> rtsp_list;
      public String current_rtsp;
      public boolean shuffle;

      private int current_rtsp_index;
    private int current_playlist_index;

      public User(String name) {
            this.name = name;
            rtsp_list = new ArrayList<String>();
            playlist_names_list = new ArrayList<String>();
            current_playlist_name = "";
            playlist_list = new ArrayList<List<Integer>>();
            current_playlist = new ArrayList<Integer>();
            current_rtsp = "";
            shuffle = false;
            current_rtsp_index = 0;
        current_playlist_index = 0;
      }

      public void add_rtsp(String rtsp) {
            if(rtsp_list.isEmpty()) {
                  current_rtsp_index = 0;
                  current_rtsp = rtsp;
```

```
            }
            rtsp_list.add(rtsp);
      }

    public void add_playlist(String playlist_name, ArrayList<Integer>
playlist, ZMQ.Socket requester) {
        playlist_names_list.add(playlist_name);
        playlist_list.add(playlist);
        if(playlist_names_list.size() == 1) {
            current_playlist_index = 0;
            change_playlist(current_playlist_index,requester);
        }
    }

    public void change_playlist(int playlist_index, ZMQ.Socket requester) {
      String playlist_name = playlist_names_list.get(playlist_index);
      List<Integer> playlist = playlist_list.get(playlist_index);
        current_playlist_name = playlist_name;
        current_playlist = playlist;

        rtsp_list.clear();

        byte[] reply = null;
        for(int i=0;i<current_playlist.size();i++){
            int temp_song_number = current_playlist.get(i);
              String request = "urls:"+temp_song_number;
              requester.send(request.getBytes(), 0);
              reply = requester.recv(0);
              if(!Objects.equals("end:",new String(reply))) {
                  String rtsp = new String(reply).replace("\n", "");
                  add_rtsp(rtsp);
              }
        }
    }

      public void go_to_next_rtsp(EmbeddedMediaPlayer emp) {
            if(shuffle) {
                  current_rtsp_index = ThreadLocalRandom.current().nextInt(0,
rtsp_list.size());
            }else {
                  current_rtsp_index++;
                  if(current_rtsp_index >= rtsp_list.size()) {
                        current_rtsp_index = 0;
```

```
                }
            }
            current_rtsp = rtsp_list.get(current_rtsp_index);

            change_rtsp(emp);
      }

      public void go_to_index_rtsp(EmbeddedMediaPlayer emp,int index) {
            if(index >= 0 && index < rtsp_list.size()) {
                  current_rtsp_index = index;
                  current_rtsp = rtsp_list.get(current_rtsp_index);
                  change_rtsp(emp);
            }
      }

      private void change_rtsp(EmbeddedMediaPlayer emp) {
            emp.stop();
            emp.prepareMedia(current_rtsp);
            emp.play();
      }

    public void go_to_next_playlist(ZMQ.Socket requester) {
        current_playlist_index++;
        if(current_playlist_index >= playlist_names_list.size()) {
            current_playlist_index = 0;
        }
        change_playlist(current_playlist_index,requester);
    }

}
```