RTP大作业报告

软73 沈冠霖 2017013569

0.TASK 1

实现逻辑和TASK 2差不多，就是客户端给服务器发RTSP请求（只实现了SETUP，PLAY，TEARDOWN三个），然后服务器解析请求。当PLAY请求发送后，服务器读取图片（实现了marker，可以读取任意大小图片）并且发送给客户端，客户端显示在label上，收到一张完整图片就显示。具体的实现方式参考TASK 2即可。

1.RTP客户端和服务器

1.1 RTSP指令的实现

实现了以下指令：

SETUP--客户端告诉服务器自己的端口，session，要播放的文件名。

GET_PARAMETER--客户端获取待播放的文件的基本信息。

SET_STARTPLACE--这个是为了实现"继续播放"附加功能设计的指令，客户端告诉服务器从哪里开始读取视频文件，因为之前的帧都已经缓存了。

PLAY--开始播放和下载

PAUSE/RESUME--暂停和继续播放，下载

TEARDOWN--停止播放，关闭客户端和连接

指令的实现都是仿照例子，客户端在RTSP连接中发送，然后服务器接收，解析，回复。

1.2 视频文件的传输和播放

服务器用opencv将视频解析成一帧一帧的图片，然后将图片发送到客户端。

客户端仿照样例，在接收到图片后将图片存储到本地，在需要播放的时候将图片显示在tkinter的label上。

控制播放速率的方式也比较简单，客户端每播放一帧视频，负责播放视频的线程就阻塞1/fps秒的时间。

1.3 数据的传输

数据使用RTP包传输，基本基于提供的RtpPacket实现，用UDP传输。

1.3.1 怎么传输较大的图片帧

首先，服务器将较大图片帧切分成一个个RTP数据包，利用包头的marker来进行传输。当marker = 0时，代表这个数据包是新的图片，否则这个数据包和上一个数据包是一张图里的。

其次，客户端解析RTP数据包，如果marker是0就增加一个帧号并且建立新文件存储，否则就附加到当前文件下。

1.3.2 怎么进行稳定传输

如果使用udp协议直接进行传输，则会有较为严重的丢包，体现在图片上，就是部分图片下边会有不正确的部分。直接传输经过测试，丢包比较严重，有不少图片有这种效果。

我的服务器实现了稳定传输，模拟的是TCP协议的GBN传输，详情不在此赘述了。

1.3.3 缓冲机制

RTP重要的机制就是缓冲，能够应对网络情况的jittering。客户端指定了缓冲的秒数，当已经接收到缓冲秒数×帧率这么多图片后才开始播放，同时保证接收帧数和播放帧数，接收线程和播放线程完全分离，这样就实现了缓冲机制。

1.4 播放器客户端的实现

播放器使用TKinter实现。

1.4.1 播放器的布局

在读取完毕视频信息（GET_PARAMETER)后，才加载播放器控件。

播放器布局用place函数，直接指定对应控件的位置。因为控件的位置依赖于待播放视频文件的大小。

1.4.2 进度条和播放速率按钮的实现

进度条使用scaler控件实现。在播放时，自动更新。进度条绑定了changeScaler函数，在进度条被人为拖动（更改过快）的时候进行处理，直接更改当前播放的帧。

播放速率使用radiobutton控件实现。

1.4.3 全屏和退出全屏

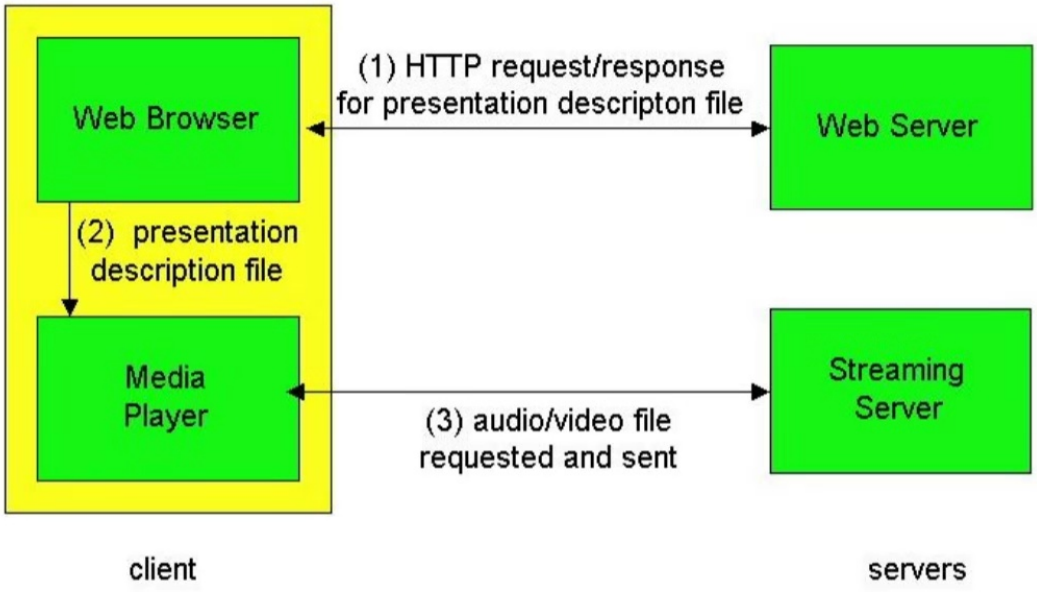
全屏和退出全屏通过左上角的最大化/恢复来实现。

代码里将tkinter的事件绑定了ChangeScreen函数。因为ubuntu有状态栏，因此我大致估计全屏后占到屏幕的96-98%之间，因此我设置屏幕宽，高都超过95%是全屏，不都超过就不是全屏。

全屏时，我设置显示图片占满屏幕，这是通过Image库的resize函数实现的。

2.主客户端和服务器

实际的服务器-客户端结构是这个图这样的。



主客户端连接主服务器，获得待播放的文件列表等信息。用户选择待播放的文件，加载RTP客户端，然后RTP客户端连接RTP服务器进行播放。

2.1 拉取播放列表，缩略图和字幕

主客户端和服务器复用了RTP和FTP的一些代码，能够实现类似FTP List指令的拉取全部视频文件的功能，还有类似FTP PORT+RETR下载单一文件功能。每个主客户端一连接，就从服务器获取视频文件列表，并且下载全部字幕和图片文件。缩略图是读取视频第一帧并且保存来实现的。

2.2 播放列表的实现

播放列表使用QListWidget实现，对每个视频，设置QListWidgetItem的icon为视频缩略图，text为名称即可。

使用QTextEdit加上字符串匹配实现了搜索。

用一个函数处理Click事件，来读取对应视频的基本信息显示在下方

用一个函数处理右键事件，弹出右键列表。

选择右键列表的选项，进入播放器客户端。

3.实现的附加功能列表，以及一些局限

一个主客户端，能够显示服务器所有待播放的视频文件名，和其缩略图，总时长，现在已经播放的时长等。但是因为时间不足，没有支持服务器里的子文件夹，以及这个播放列表是静态的，中途修改服务器会出错。

主客户端可以搜索想看的视频。

主客户端可以实现继续播放---在主客户端未关闭的前提下，这次播放一个视频到某一位置退出，可以继续从这一位置播放。

播放器客户端实现了全屏功能，但是全屏后的帧率有所变化，不一定完全吻合，因为opencv的运算占用了时间。

播放器客户端实现了字幕，可以自己判定是否有字幕文件，并自动播放，但是只支持了src格式。

播放器客户端实现了buffer机制。

播放器客户端和服务器实现了GBN稳定传输。

4.一些难点和总结

4.1 线程问题

这个大作业的线程关系比较复杂，具体如下。

服务器主线程每接收到一个客户端连接，就另开一个Manager线程。一个Manager线程和一个客户端一一对应。

客户端主线程负责响应tkinter界面，发送RTSP请求。另开一个线程，无限循环接收RTSP回复并处理。

在客户端SETUP后会开一个线程负责接收服务器RTP数据。在接收数据达到一定程度，即填满buffer后，会开启一个线程专门负责更新播放图片。

服务器Manager主线程负责接收并处理，回复RTSP请求。在PLAY指令后会开一个线程负责读取和发送图片数据。

线程间通过各种对象的全局参数来通信，避免了各种冲突的存在。

4.2 怎么传输较大的图片帧

首先，服务器将较大图片帧切分成一个个RTP数据包，利用包头的marker来进行传输。当marker = 0时，代表这个数据包是新的图片，否则这个数据包和上一个数据包是一张图里的。

其次，客户端解析RTP数据包，如果marker是0就增加一个帧号并且建立新文件存储，否则就附加到当前文件下。

4.3 总结

通过这次大作业，我更加深入理解了RTP主，播放器客户端，服务器的机制，以及多线程的实现方法。还对两个图形化界面--tkinter，pyqt有了更深入的了解。还有，通过这次实验用UDP传输大量文件，我切真体会到了丢包的严重，并且尝试用GBN解决了丢包问题。FTP打下的对socket和多线程的理解很关键，也同样要感谢助教提供的样例代码，让我入门比FTP容易许多。