大作业part3 实验报告

2023年5月31日
小组成员：
林欣涛 2020012356
张雨恬 2020012392
汪静雅 2020012375

一、实验内容

实验背景

本次实验旨在在之前两次实验完成的ALSA音乐播放器基础上，扩展音乐播放器的功能。我们通过重载QListWidget组件与设计一系列逻辑控制，重写了播放、暂停、继续和停止等功能。同时，我们利用重载QListWidget组件实现上一曲和下一曲功能，通过切换文件读取头实现快进和快退功能，并利用抽帧和插值技术实现倍速播放功能。此外，我们还需要解码转换MP3格式音乐并进行播放，并利用QT框架实现在开发板上启动的GUI界面。

实验前置知识

1. 音频编码和解码：音频编码是将原始音频信号转换为压缩格式的过程，常见的编码格式包括MP3、AAC等。音频解码是将压缩格式的音频解码还原为原始音频信号的过程，以便进行播放或处理。
2. GUI库：GUI（图形用户界面）库是用于创建图形化用户界面的软件库。在项目中，需要选择一个适合的GUI库来创建音乐播放器的界面，例如Qt、SDL等。
3. 网络连接：在项目中，需要通过网络连接开发板或其他设备。了解如何建立和管理网络连接以及网络传输的相关知识对于实现项目功能是必要的。
4. 声音播放：了解如何在计算机上播放音频是必要的，包括使用音频库或API进行音频输出，例如使用ALSA（Advanced Linux Sound Architecture）库来控制音频设备和播放音频文件。
5. 音频格式转换：在项目中，需要将不同的音频格式进行转换，例如将MP3文件转换为WAV文件。了解音频格式和音频转换的相关知识，如采样率、位深度和声道数等，对于处理音频文件是重要的。
6. 时间拉伸算法：在实现变速播放功能时，会涉及时间拉伸算法。了解不同的时间拉伸算法，如抽帧和插值算法、短时傅里叶变换等，以实现音频的变速播放效果。
7. 多线程编程：在项目中，需要使用多线程来处理耗时的操作，如音频解码、文件转码等。了解多线程编程的概念和相关技术，如线程的创建、同步和通信，可以提高程序的效率和响应性。

实验目标

本次实验的目标如下：

1. 实现上一曲和下一曲功能的调节。
2. 扩展音乐播放器以支持mp3格式音乐的播放。

3. 实现多种倍速播放功能（0.5倍、1倍、1.5倍和2倍），要求变速时音调不变。
4. 实现快进10秒和快退10秒功能。
5. 扩展功能：自行设计GUI界面，并在开发板上实现上述基本功能的GUI界面化。

二、实验部署

部署环境介绍

- 硬件环境
  - 电脑系统
    - 本实验使用到的电脑系统为Windows 10、Windows 11和macOS。
  - 开发板系统
    - 系统采用MPU和MCU双平台设计，MPU用于Linux操作系统开发，MCU用于ARM体系结构与裸机编程。
    - 采用高性能多核异构系统架构芯片STM32 MP157A，包含双核Arm Cortex A7和单核Arm Cortex M4。A7内核应用处理器主频为650MHz，M4内核微控制器主频为209MHz。
    - **音频编解码芯片**：采用WM8960这款高度集成的低功耗立体声编解码器。WM8960集成了音频输入、输出、麦克风预放大器、耳机驱动器等功能，同时提供I²S（Inter-IC Sound）接口与控制设备通信。支持多种采样率和音频数据格式，以实现高质量音频处理。
- 软件环境
  - xshell版本: Xshell 7(Build 0122)
  - xftp版本: Xftp 7(Build 0119)
  - vmware版本: 16.2.2 build-19200509
  - ubuntu版本: 18.04.6 LTS
  - QT版本: 5.14.1

硬件连线实物图

三、实验过程

实验操作

由于本次实验需要 使用QT完成附加功能GUI界面 ，因此需要先对前两次实验完成的代码结构进行重构，将其项目化、模块化，方便后续功能的添加以及前端界面的调用。

随后，我们依次完成了QT界面设计与槽函数的连接，文件导入，重新设计播放、暂停、继续、停止功能的逻辑，解码转换MP3格式音乐，重载QListWedgit组件实现歌曲切换逻辑，修改线程逻辑实现线程锁，实现音量调节与界面滑条之间的同步，通过修改文件读取头实现快进快退，通过计时器播放进度条逻辑设计与播放结束逻辑，通过抽帧和插值技术完成倍速播放功能。**总计完成代码量接近1200行，这里统计的行数不包含引用的minimp3.h头文件与QT的ui文件与资源文件**。

下面详细介绍每个部分的实现逻辑：

- 1、**QT界面设计与槽函数的连接**：这个部分的主要是回顾大一暑假小学期的QT知识。首先是需要考虑开发板的长宽比例(测量约为15.5cm:8.8cm)来 提前固定QT界面的大小 ，这里使用的是800:460(这里是一个近似的比例，主要时方便元素排版设计)，能够避免设计元素在开发板上的显示变形问题。同时，其他的前端工作主要是 控件的排版 以及相关图片、图标等的插入，一些次要的例如 所有的按钮都添加了悬浮动画 、 重新设计标题栏 等这里不赘述。除此之外，还需要对每个控件的槽函数进行连接，这里我们使用的是 QT的信号与槽机制 ，将每个控件的信号与对应的槽函数进行连接，实现控件的功能。下图即为我们所设计的页面ui：

其中播放栏中的按钮从左到右依次为 `后退10s`，`播放暂停`，`快进10s`，`停止`，`导入歌曲`，`音量选择`，`倍速选择`

其中播放列表右侧的按钮从上到下依次为 `上一曲(向上箭头)`，`下一曲(向下箭头)`

- 2、文件导入：使用QFileDialog打开文件对话框，将选取的文件信息插入到QListWidget控件当中，这里通过重载QListWidget组件，在其构造函数当中读取、存储文件的路径、文件名等基础信息，同时将信息展示在播放列表当中。
- 3、重新设计播放、暂停、继续、停止功能的逻辑：
  - 这个部分的前端显示逻辑通过变量 `playing` 和 `播放进度条的数值` 两个部分联合判断播放情况：对于playing为true的，调用播放逻辑的pause，对于play为false情况，如果进度条数值为0表示是新音乐，调用播放逻辑的play，不为0表示是之前被暂停的，调用播放逻辑的continue。除此之外，通过对于QListWidget组件的重载，给每个item添加上播放的标记信息，用于在播放列表显示乐曲的播放状态。
  - 对于这个部分的播放逻辑采用的是大作业Part2当中的播放代码，通过将这部分的代码迁移到QT当中，同时分成 `读取文件头open_music_file`、`初始化pcm设备init_pcm` 以及 `播放play` 三个模块，这里通过使用 `playing` 和 `isplaying` 两个变量分别表示 `是否播放了音乐(用于开始和停止)` 和 `是否正在播放音乐(用于暂停和继续)`，随后在pause、continue和stop三个函数当中进行修改这两个播放状态来完成音乐的暂停、继续和停止的功能。
- 4、**实现歌曲切换逻辑**：这个部分是建立在前面的播放和停止的逻辑以及QListWidget组件之上
  - 在前端的显示上，通过 `设置QListWidget的currentRow参数`，加一减一以及边界判断，即可在播放列表显示上实现歌曲的切换，最后一首歌往后切会回到第一首歌的 `循环切换`。除此之外，`我们还实现了通过双击QListWidget中单个Item项的方式来进行歌曲的选择，不局限于一首一首的切换播放`，更加的便捷。
  - 在播放的逻辑上，清除当前歌曲的播放状态，通过getcurrentRow获取现在需要播放的新歌曲，调用播放按钮的槽函数即可实现播放
- 5、**解码转换MP3格式音乐**：

    - 这个部分使用了minimp3这个单文件的音频转码库，虽然这个库可以将MP3直接转化为PCM设备能够接受的字符流，但是考虑到WAV和MP3需要复用一样的播放代码，所以这里采取的方式是使用minimp3先 `将MP3文件转化为临时的WAV文件` ，再调用前面的播放函数进行播放
    - 这个MP3转成WAV的实现逻辑主要是学习了博客园的相关文章，将在minimp3的github中minimp3.h文件下载并放进QT项目当中，随后在项目当中实现对于获取到的MP3文件，使用mp3dec_decode_frame函数读取MP3的文件头信息，使用realloc函数读取其文件数据流信息，再将这些信息根据WAV文件头的格式依次写入到临时的.temp.wav文件当中
- 6、修改线程逻辑实现线程锁：这个部分是因为在大作业的Part2当中，音乐的播放是在主线程完成，而音量调节在新开的线程当中完成，但是这并不适用于QT当中，因为主线程需要考虑UI的互动，以及调节播放状态的也不只有音量，所以将这个部分修改为 `每次播放音乐都会新开一个线程用来播放` ，其他修改状态的函数在主线程完成。同时，考虑到多线程的线程安全与资源竞争问题，因此对于显示当中将playing参数，在播放逻辑部的playing和isplaying以及从wav文件读取数据等部分加上了线程锁， `避免多线程同时访问同一资源导致的错误` 。
- 7、实现音量调节与界面滑条之间的同步：
    - 这个部分的显示逻辑，只有在程序开始是读取Player的volume信息进行初始化显示，随后的音量调节部分都是通过对于QSlider的调节触发设置的槽函数，槽函数当中直接获取QSlider的value，随后调用播放逻辑当中的音量设置。这里通过 `使用QSlider组件实现了音量调节的连续性` 。
    - 在播放逻辑上，这里迁移了在大作业Part2当中音量调节代码，但是考虑到这里使用了QSlider进行调节，所以删除了原先代码对于按键部分读取，同时将其拆分为 `初始化混音器init_volume` 和 `音量设置mp3dec_decode_frame` 两个部分。这里就是通过拿取QSlider的value数据，通过百分比计算当前的实际音量值，随后使用snd_mixer_selem_set_playback_volume_all函数进行音量设置
- 8、**`通过修改文件读取头实现快进快退`**：
    - 这个部分的前端逻辑很简单，点击快进或者快退按钮，直接调用播放逻辑当中的快进与快退函数，随后设置将当前的时间进度设置跳转到对应的数值
    - 在播放逻辑部分，这里使用了 `10 * rate * num_channels * bits_per_sample / 8` 的公式来计算10s内正常速度读取的数据量，随后使用 `ftell` 函数获取当前文件指针读取的位置，返回值是数据量，随后简单判断一下快进和快退的边界情况， 确定下来需要切换的数量量，使用 `fseek` 函数设置文件指针的新的读取位置。这个时候播放函数在再次从文件中读取buffer的时候就会从新的文件指针的位置进行读取，从而实现了快进与快退。
- 9、通过计时器播放进度条逻辑设计与播放结束逻辑：这里的主要工作是获取音乐的总秒数，随后通过当前播放的秒数去计算时间进度条的进度，然后将两个总秒数转化为时间格式显示出来。
    - 总秒数的获取，是在打开音乐文件的时候，通过 `fseek(fp, 0, SEEK_END);` 将文件指针放在文件的末尾，随后使用 `fsize = ftell(fp);` 读取到文件总的大小，再使用公式 `fsize/(rate * num_channels * bits_per_sample / 8);` 用总的文件大小除去每秒钟处理的数据量，就可以得到总的时间。
    - 对于当前的播放的秒数的获取，在每次音乐开始播放的时候将会初始化当前时间为0s，随后开启计时器，设置每1s触发一次时间函数，在时间函数当中，当前的秒数每次添加上播放的倍数speed*1，作为最新的秒数

    - 播放结束逻辑是当当前播放的秒数大于总的秒数的时候，调用停止按钮的槽函数，终止播放。
- 10、**通过抽帧和插值技术完成倍速播放功能**：倍速播放这个部分单纯通过修改采样率会导致音调随着变化，而实现或者调用短时傅里叶等高级时间拉伸算法又过于复杂，因此这里使用效果相对粗糙的抽帧技术实现大于1的倍速，使用插值技术实现小于1的倍速。
    - 抽帧技术的实现，这个部分的大概思路就是将需要输入PCM设备的播放数据进行切分，然后删除其中的部分数据。考虑到实现的复杂度，这里采用的是相对简单的`没有平滑处理的等距抽帧`的方式，于是引入了两个调整参数——k是切分的数量，m是相邻删除区域的间隔。在调整k和m的过程当中，有如下发现：m不变的情况下，不断增大k的值，会出现音质先急速下降，又缓慢回复的过程，同时伴随着音调从一开始的基本正常到k接近rate之后变得越来越高；k不变的情况下，不断增大m的值，可以发现音调基本没有变化，但是音质在缓慢地提高。这个过程可以通过音频波的形态进行理解，由于k值的增大，一个周期内数据被删除的越多，处理一个周期的时间长度变小，相对于的频率、音调就会变高；对于m值的增大，被删除的块更加集中，音频波的断裂数量相对越来越少，由此产生的杂音就相对较少。
    - 插值技术的实现，这个部分的大概思路就是需要将输入PCM设备的播放数据进行细分和扩充。考虑到实现的复杂度，这里采用的是相对简单的`没有平滑处理的克隆`的方式。同样的引入了两个调整参数——k是切分的数量，m是相邻扩充区域的间隔。和抽帧相似的规律，在m不变的情况下，不断增加k，音调从开始的不变，最后变为低音; 在k不变的情况下，不断增加m，音质的杂音也会相对减少。
    - 考虑到上述量种情况相似的规律，于是K需要尽可能取得小，k_per_size * m则需要尽可能取得大。为了进一步扩大 k_per_size * m的值，这里还将周期的大小增加了1倍，以取得相对更优的倍速效果。

实验中遇到的问题与解决方法

- 1、GUI库的选择：由于选择完成额外功能实现GUI界面，而播放代码需要与GUI界面进行适配，所以需要优先选择GUI库，这里尝试了包括SDL2(支持窗口绘制的音频播放库)、cursor(支持终端文本窗口绘制)等在内的各种GUI库，主要有几个困难点——类似于SDL2库这样的便捷的界面创建库在交叉编译连接当中并不存在，给交叉编译链添加配置新的工具库过于的复杂；而类似于cursor库这种文本窗口，是一种显示在终端的ui界面，不符合显示在开发板上的要求。

解决方法：助教提供了《如何在交叉编译当中完成QT程序的编译工作》的教程，以及百科荣创的虚拟机当中也包含有QT工具，于是最后决定使用QT作为gui的最终选择。

- 2、打开xshell后再连接网线，此时xshell疑似进入网络传输模式，xftp通过网线无法连接开发板

解决方法：先连接网线，再打开xshell

- 3、解码MP3格式音乐：这个地方和GUI库的选择部分遇到了类似的问题，因为使用ffmpeg这个较为成熟的音频转化库也会遇到交叉编译链当中不存在这个工具需要自己手动配置进去的问题。

解决方法：这里通过在网上查阅单头文件转化MP3的开源仓库，谷歌推荐了minimp3这个仓库，通过简单的阅读和资料的查阅，这个minimp3不需要额外的库环境，可以通过交叉编译，所以最后选这了minimp3这个音频转化库

- 4、QT编译需要alsa库：在将播放的部分代码迁移到QT后，点击编译一直报错，显示无法使用ALSA库的函数文件。

解决方法：这是因为编译指令需要带有-lasound，这个需要在QT的pro文件当中配置，指定链接alsa库的选项，例如pro文件最后添加上LIBS += -lasound即可解决问题。

- 5、获取音乐播放的总时长：虽然正确使用了 `音频数据大小/(rate * num_channels * bits_per_sample / 8)` 这个公式，但是在辨别谁才是真正的音频数据的大小的时候，遇到了很多的问题，尝试了sub_chunk2_size、buffer_size等数据都显示不正确。

解决方法：这个地方需要获取的音频数据其实是整个文件的数据大小，这个可以从播放音频的时候使用fread的读取函数可以观察到，于是查阅资料找到可以读取整个文件长度的实际大小的操作 `fseek(fp, 0, SEEK_END);` 获取文件头指向文件末尾， `fsize = ftell(fp);` 获取以及读取的数据，这样返回的数据就是整个文件的数据。

- 6、倍速技术选择：在倍速上先是尝试了调解采样率的操作后发现音调变化过大，使用密集抽帧和密集插值的方式也没有太差差别，音质反而还差了。所以这个问题花费了很长的时间去调研，了解到想要实现变速不变调，还不损音质，谷歌推荐可以考虑使用卷积插值、短时傅里叶等时间拉伸算法，但是这些算法相关的论文虽然是开源的，但是实现这些算法的代码并没有开源，而已经实现了这些时间拉伸算法的成熟音频库，却因为不在交叉编译链当中而无法使用。而直接从论文复现代码难度过大。

解决方法：这里最后还是采用了抽帧和插值的方式，但是相比最初的尝试，这里的实现给其中加入了调整参数，花费了一个下午的时间调参炼丹出来了一个相对较优的结果，能够做到音调不变，杂音相对较少。

四、实验结果

实现结果展示

- 虚拟机当中使用需要将player.h文件当中的card初始化为“default”，selem_name初始化为“Master”，再进行编译。
- 如果是需要编译到开发板上运行的程序，则应该将player.h文件当中的card初始化为“hw:0”，selem_name初始化为“Playback”，再进行编译。
- 编译方式与执行QT可执行文件的方式

```
// 编译QT项目
source /opt/st/myir/3.1-snapshot/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
qmake musicplayer.pro
make

// 将可执行文件转移到QT上执行
kill mxapp2
chmod +x musicplayer
./musicplayer
```

实现效果参考录屏文件 `demo.mp4`

实验优化

在实验过程中，我已经完成了一些优化，但由于时间有限，还有一些优化点暂未处理。

1、导入文件预转码MP3文件：这是因为在播放MP3前需要先将其转码为WAV，相对比较耗时，大约需要等待1s左右的时间，这个部分可以考虑在导入文件的时候，开一个新的线程，给每个MP3先预转码一个WAV文件，实现用空间换时间。

2、音乐播放的时间进度：由于计时器每秒更新一次进度，所显示的时间进度可能与实际播放进度存在0.5-1秒的误差。因此，拖拽进度条切换进度可能会有较大误差。优化思路是增加计时器的更新频率，缩小更新时间间隔，以尽可能减小误差。然后启用进度条的编辑功能，类似于快进和快退的方式实现自由的进度修改。

3、支持更多的音频播放格式：目前使用的minimp3库仅支持MP3文件的解码转换，对其他文件不支持。可以学习交叉编译环境配置后，考虑使用成熟的音频库或直接使用目标播放格式的单头文件库。

4、使用更加完善的时间拉伸算法：针对当前的抽帧间隙进行优化，减少杂音。例如，在实现2倍速时，将相邻帧重叠50%，在重叠处进行短时傅里叶等平滑处理，减少音频失真。对于插值的扩展，可以考虑使用卷积插值等高级方法，将相邻帧的前50%和后50%进行平滑扩展，然后进行插值处理，以减少音频失真。

5、UI上的CD机图片可以新增旋转动画设计：这个部分在代码上的实现采用的是QPixmap格式的图片进行设置，QPixmap格式的图片可以设置一个transformed的旋转操作，只要给播放进度条设置一个监督触发函数，播放的时候触发设置QPixmap进行旋转就可以实现播放时候的一边旋转的CD机动画

五、实验心得

在这次实验中，我遇到了一些挑战，但也学到了很多知识和经验。通过实践和探索，我对音乐播放器的实现有了更深入的了解，并提高了我的编程能力和解决问题的能力。我学会了选择合适的GUI库，并解决了与GUI界面适配的问题。我使用minimp3进行了MP3格式音乐的解码，同时获取音乐播放的总时长。另外，我尝试了不同的倍速技术，虽然最终采用了抽帧和插值的方式，但我也了解了其他时间拉伸算法的原理。在实现过程中，我还使用了多线程来提高程序的效率和响应性。

通过这次实验，我不仅掌握了实现功能所需的技术和知识，也培养了解决问题的能力和灵活应对挑战的态度。我相信这些经验和收获将对我的未来学习和工作产生积极的影响。我深刻体会到，解决问题的过程并不总是一帆风顺的，但通过持续的学习和努力，我们可以克服困难，实现自己的目标。这次实验让我更加自信地面对编程任务，并激发了我继续深入学习和探索的热情。

六、参考链接

- 1、mp3格式转wav格式 附完整C++算法实现代码 - cpuimage - 博客园 (cnblogs.com)
- 2、lieff/minimp3: Minimalistic MP3 decoder single header library (github.com)