xv6 音乐播放器之
实现与改进

刘峻琳 彭友 李映辉 刘博格

2015.1.23

1 总述

在本次实验中，我们以上届学长代码为基础，力求在 xv6 中实现音乐播放器的功能。上一版的代码修改了 xv6 操作系统的文件系统，使之支持足够大的音乐文件，此外还实现了声卡驱动。

然而上一版的代码仍有如下不足：WAV 格式音乐只能在高配电脑上流畅播放、MP3 根本无法播放（存在较多 BUG）以及没有真正实现多进程调度。我们的目标就是在这几个方面对播放器做出改进。

2 实验环境

一级虚拟机：Ubuntu 12.04 或 10.04（不支持 14.04）

二级虚拟机：QEMU

编译器：GCC 4.4

3 主要 BUG 的排查和消除

MP3 是一种高压缩率的有损压缩音频格式，为了播放 MP3，我们需要对该格式的数据文件执行一套非常繁锁的解码流程（即解压缩），将之解为原始 PCM 数据送入声卡。一旦这个部分正确实现，MP3 的播放就解决了百分之九十。

3.1 MP3 解码算法的正确性验证

MP3 解码算法非常复杂，我们沿用了上一版代码中引入的开源代码。由于上一版的 MP3 根本无法播放，因此我们首先需要验证这一套开源代码的正确性。我们找到了该开源代码的 windows 版本，在 windows 下对 MP3 文件进行了解码，将解码得到的 PCM 数据导入 Audacity 软件，声音如期播放，因此该开源代码的正确性得到验证，接下来我们要做的是就是让它在 xv6 上跑起来。

3.2 改进代码使之适应 xv6 的用户栈大小

在上一版的 xv6 中，遇行 MP3 播放进程会遇到系统抛出的异常，如果使用 printf 进行单步调试，则 printf 的数量和位置将会对系统行为造成影响，具体体现为 printf 会随机输出一些乱码并且不同的 pritnf，系统抛出的异常会不同。经过不断地分析思考和查阅 xv6 官方文档，我们发现并确定造成该问题的原因是 xv6 的用户栈大小只有一页大小（4KB）且不能动态增长，而整套解码流程所需要的栈开销远远大于 4KB，主要体现在多处局部二维数组的定义上。因此，我们

小心翼翼地修改整个算法的代码，尽最大可能减少其对栈的利用。完成该项工作后，系统异常没有了，但是我们并没有得到正确的解码结果。

3.3 GCC4.4 不支持 bool 类型

在整个项目中，我们使用的编译器版本是 gcc4.4，是一个非常早的版本，不支持 bool 类型的变量。因此，代码采用的办法是用宏定义的 TRUE 来代替布尔值，将 TRUE 定义为 1。这就带来了问题，在有布尔值的情况下，任何非 0 的整型数和 true 都是等价的，然而和宏定义的 TRUE 并不等价。但是在解码流程中，有大量用非 0 整型数和 TRUE 作比较的地方，因此会带来一些错误结果。尽管找到这个问题并不容易，但是解决它却非常简单，只需直接将非 0 数作为条件判断句就可以了，不需要将它和 TRUE 作比较。

3.4 变量名混用

上一版的代码对解码算法稍稍做了修改，在修改的过程中出现了变量名混用的情况，我们对这样的情况做了修改。

3.5 GCC4.4 对指针的用法异常

在代码的某些地方，我们发现 gcc4.4 对某些空指针和 NULL 的比较出现了问题，指针明明是空的，但是和 NULL（宏定义为 0）比较时却判断为不相等，因此造成了代码执行流程出现了混乱。对于这类情况，我们同样小心翼翼地修改了代码，但却保持了原有算法逻辑，从而解决了问题。

4 数学函数的实现

MP3 的解码算法需要用到大量浮点数的数学运算，然而 xv6 里并没有任何现成的数学库的支持。如果引入 C 的标准库，则其相互依赖太多，引进难度非常大。而自己手动实现数学函数根本无法达到需要的精度和速度。

为此，我们翻遍了互联网上的资源，终于找到了一些可用的开源代码，再加上自己写的一点代码，提供了一套用内联汇编实现的数学函数。

对这套数学函数，我们必须对它进行足够的精度和速度测试。我们用千万数量级的测试数据进行了测试，该套数学函数精度可以达到 10-12，速度大概为标准库的 1.5 倍左右，已经达到了可以用的性能。因此，我们可以放心地引入。

5 多级缓冲的多进程播放

在解决了以上诸多难题后，MP3 的音乐终于可以播放出来了！然而目前的播放仍是单进程的，我们需要实现真正的多进程播放。为此，我们设计了多级缓冲的多进程播放机制。实现了多级缓冲之后，WAV 格式的音乐在一台低配电脑上也能流畅播放了！

5.1 一级缓冲：共享内存

一级缓冲实际是共享内存的实现，读取文件的进程不断从数据文件一帧一帧地读取音乐数据，并将其送入共享缓冲区队尾。解码进程不断从队首读出一帧数据，对其进行解码操作。同时为了避免读文件的进程执行得过快，我们设置了一个阈值 1000，即读文件的速度不能比解码的速度快过 1000 帧，否则将无法有足够大的缓冲区装下数据。同时，共享内存就会涉及对临界区的访问，我们也实现了互斥锁来保护临界区数据。

5.2 二级缓冲：缓存播放

二级缓冲同样也是用队列实现，解码进程不断将将码结果送入待播放队列，而播放的进程则不断从该队列读取固定大小的块数据，并将其送入声卡。有一个 4096*8 字节大小的缓存区作为循环队列，头指针 head（代码中用变量 in 表示）指向，尾指针 tail（代码中用变量 out 表示）分别指向了当前读入读出的起始字节。读入的时候只要不超过剩余缓存大小都可以，读出是以 4096 字节为一个块进行操作。这样读入和读出就实现了分离，从而起到了 IO 繁忙和 CPU 繁忙的进程独立运行的局面。

6 总结

综上所述，在我们的实验中，我们完成了如下工作：

- 解决了 MP3 在 xv6 上播放遇到的诸多问题，使之达到接近流畅的播放程度
- 实现了真正的多进程协同播放
- 实现了多级的缓冲机制，使 WAV 格式的音乐可以在一台低配电脑上播放

7 备注

我们的代码是基于上一版代码进行修改的，读者如对某些技术问题而无法在本文档找到答案，可以查阅王思伦《操作系统实验文件系统报告》。

8 附：文件说明

playmp3.c 读取 MP3 音乐进程代码

mp3dec.c 解码进程代码

decodemp3.c 解码核心函数

common.c 解码需要的一些辅助函数

huffman.c 哈夫曼解码的核心函数

sound.c 声卡操作

sysaudio.c 进程通信核心代码