

# XV6 实验报告

## 基于xv6的mp3音频解码与播放

卢凌铜 肖剑楠 董亮 杨志强 - January 23, 2015

```
QEMU
init: id isn't equal to ioapicid; not a MP
starting
starting sh

1 1 512
1 1 512
2 2 1929
2 3 10544
2 4 10017
st 2 5 6545
2 6 11888
2 7 10346
2 8 10005
2 9 9999
2 10 11711
2 11 10078
2 12 10067
2 13 18479
s 2 14 10549
sts 2 15 40460
2 16 10882
2 17 9791
e 3 18 0
```

## 导语

本次实验，我们小组在0字班与1字班的基础上。进一步实现了xv6平台上，对于mp3文件的解码与播放。

关于音频的播放，主要分为以下几个模块

- 硬件初始化
- 文件系统修改
- 播放逻辑与进程通信

其中，前两个模块的工作几乎完全基于前两届学长的工作。在这里做简要陈述：

1. 硬件初始化 绝大多数笔记本的声卡硬件使用的是Intel旗下的ac97声卡。为了使xv6能够正常调用硬件声卡。首先我们需要在qemu虚拟机里面实现其声卡驱动。具体过程时找到相应设备的PCI总线地址，找到ac97相应的位置。最后在Makefile里添加相应的虚拟声卡代码段，即可成功配置。详细流程可参见上一届文档，这里不再赘述。

```
QEMUOPTS = -soundhw ac97 -hdb fs.img -hda xv6.img -smp ${CPUS}
```

2. 文件系统 原生xv6文件系统只支持很小的文件。这里需要对其文件系统做比较系统的修改（具体修改见fs.c 以及 mkfs.c ）。另外，在需要添加某个文件时，只需要在Makefile里做相应的添加即可。详细的修改流程请见上一届文档。

```
fs.img: mkfs README $(UPROGS)
./mkfs fs.img README in.mp3 output.pcm dewindow.txt huffdec.txt $(UPROGS)
```

接下来的播放逻辑与进程控制将在接下来几章里做比较详细的讲解。我们将从mp3解码、播放两个模块讲解。（值得注意的一点是，由于本组作业不涉及wav文件的播放。在我们的代码中移除了wav播放的相应代码段。具体wav播放及其流程可以观看上一届学长的文档和代码）

## mp3解码

在说到mp3解码时就必须谈及mp3编码的相关知识。（相关mp3解码的知识见我的印象笔记<https://app.yinxiang.com/l/ACRp1caV3s1PSJttjR14R2HvCHIQXysNN4Q>）

mp3编码的中心思想是人耳听觉范围局限在频率为20Hz到20kHz区段。而音频系统另一个重要的特点是某些音频信号会“掩盖”其相邻信号，我们就可以通过移除音频文件中一些对最终听觉效果没有影响或影响不大的音频信号，最终实现压缩文件体积的目的。

介绍完编码之后进入解码。而进入解码的第一件事就是将文件从物理帧转换为逻辑帧：

```
00000000  FF BF A7 FF FF FF FF FF F9 22 BB 44 A0 A9 D5  11111111 11111111 11111111 11111111
00000001  BA C3 B0 54 15 72 C1 50 55 AB 76 30 AB 75 98 FD  11111111 11111111 11111111 11111111
00000002  CA 3E 50 00 00 01 52 E3 48 58 40 2A DA E2 89 FB  11111111 11111111 11111111 11111111
00000003  76 C4 94 2E E6 10 69 1B 4C 1B 51 BD 69 3A 07 75  11111111 11111111 11111111 11111111
00000004  DB 51 7E 11 A3 F9 08 6A 57 A4 5E 8C 30 DA 57 30  11111111 11111111 11111111 11111111
00000005  13 94 AA 3D 24 D1 25 89 ED 1C 14 10 9D 32 1F 51  11111111 11111111 11111111 11111111
00000006  32 11 21 30 C8 9D C2 AB 45 D9 25 43 AA 42 20 19  11111111 11111111 11111111 11111111
00000007  C4 EC 6C 40 E1 34 16 59 68 59 E9 45 99 C6 5B 95  11111111 11111111 11111111 11111111
00000008  2B 56 12 0A 64 D9 A6 0E 6F F3 71 9F 36 9B 7B 69  11111111 11111111 11111111 11111111
00000009  13 39 69 1A 1B AB EA A6 DF DB B5 33 C3 84 1A 59  11111111 11111111 11111111 11111111
0000000A  81 26 9A 53 A3 6D 2A 83 B7 6D 8B 23 A6 E2 8A F3  11111111 11111111 11111111 11111111
0000000B  93 EB 0B 2C E9 AF A4 B2 98 AA A7 54 DC 61 7B 6D  11111111 11111111 11111111 11111111
0000000C  2D AD 2D 99 F6 DF 71 B7 6E 6B 01 63 A4 E7 32 4D  11111111 11111111 11111111 11111111
0000000D  FD 84 78 58 0F 49 DB B6 DE EC 67 0F 6C D4 08 00  11111111 11111111 11111111 11111111
0000000E  00 00 30 07 E3 B0 E1 5A 71 FC 66 F9 70 C0 32 0B  11111111 11111111 11111111 11111111
0000000F  C0 75 7F 7A AA B5 48 CA 21 94 66 B2 CF 11 99 A8  11111111 11111111 11111111 11111111
00000010  59 4F AA F9 87 2A FF FE FE 7E 48 46 50 29 DC  11111111 11111111 11111111 11111111
00000011  30 72 68 3C 61 A4 FF FE 78 64 7A 86 05 D2 76 5A  11111111 11111111 11111111 11111111
00000012  51 E9 35 B6 30 82 4B 4E 3D 82 18 16 EE 11 63 A7  11111111 11111111 11111111 11111111
00000013  B0 D6 98 C5 89 EC F8 C0 99 98 6F 12 65 D2 78 5D  11111111 11111111 11111111 11111111
00000014  05 16 82 5C 56 5C 98 A3 EE 32 C0 C0 00 90 E2  11111111 11111111 11111111 11111111
00000015  95 5C 9F 45 30 3D 2F E8 4A 00 92 85 FB 8B F4 C6  11111111 11111111 11111111 11111111
00000016  23 6F D4 14 B8 50 BF B6 27 D4 17 FE E4 05 05 41  11111111 11111111 11111111 11111111
00000017  38 DA F1 92 D1 AB C7 DE 59 EC 19 BE 67 96 E5 68  11111111 11111111 11111111 11111111
00000018  F2 0C 11 46 4A 49 24 EA 85 2E 2C A1 51 0A EA 58  11111111 11111111 11111111 11111111
00000019  66 7F 64 5F 57 BF 1E F2 66 B5 91 38 B8 D7 49 SE  11111111 11111111 11111111 11111111
0000001A  BD FD 6B EC FE 6D 0D E1 CB C6 B9 12 8C EA 1A FB  11111111 11111111 11111111 11111111
0000001B  5D AB E4 B7 96 6A FF 43 9A 5E BA 19 1E 8D 49 0D  11111111 11111111 11111111 11111111
0000001C  28 E9 53 CB 15 5F 7F DF 73 CB 64 65 E1 EF E1 89  11111111 11111111 11111111 11111111
0000001D  24 1B 50 D3 2A EB EE 18 81 7D FE D2 5E DF C2 FF  11111111 11111111 11111111 11111111
0000001E  98 ED 93 B8 C5 2A FE 8D 29 20 88 00 02 01 00 0E  11111111 11111111 11111111 11111111
0000001F  D4 84 C6 8C 5D 36 B2 2B 53 1D FD 1B D9 07 93 9C  11111111 11111111 11111111 11111111
00000020  F2 0C 11 46 4A 49 24 EA 85 2E 2C A1 51 0A EA 58  11111111 11111111 11111111 11111111
00000021  55 0A 20 28 FF D9 5F AE 9A 38 40 00 00 01 4E D2  11111111 11111111 11111111 11111111
00000022  F5 48 A3 8F C7 25 FC 58 5C A2 44 12 67 56 5D 72  11111111 11111111 11111111 11111111
00000023  B2 53 C5 17 1D 3B 82 A7 2B F8 02 64 F1 95 B3 65  11111111 11111111 11111111 11111111
00000024  98 B6 19 CC D3 FE 96 D1 01 5C 23 A8 B4 01 76 3A  11111111 11111111 11111111 11111111
00000025  4B 73 64 36 55 D3 3A 53 49 76 55 E9 29 2B 9C B2  11111111 11111111 11111111 11111111
00000026  B6 E0 8D 42 8E E9 C9 53 46 2A 42 7A B7 24 45 CD  11111111 11111111 11111111 11111111
00000027  6E FD 8D 72 78 51 56 95 53 E2 5A D8 F9 98 D2 FF  11111111 11111111 11111111 11111111
00000028  7B 3F D5 AD C8 63 FB EE FD 19 B8 27 39 07 18 35  11111111 11111111 11111111 11111111
00000029  3A 1B EC 5E 26 BA 73 FE 27 AF FD 92 36 9C 44 22  11111111 11111111 11111111 11111111
0000002A  D5 B2 F7 AD C8 57 F2 C7 0C 7C AF 94 4B E5 1D 6E  11111111 11111111 11111111 11111111
0000002B  BE 37 0C 50 90 24 12 06 24 5E 1E 28 79 5C D4 24  11111111 11111111 11111111 11111111
0000002C  BD 67 28 EE 8B A5 FF FE 78 64 5E 86 06 11 74 74  11111111 11111111 11111111 11111111
0000002D  53 0F 35 36 39 84 9B 4F 30 27 58 1B F1 FD 4F 4D  11111111 11111111 11111111 11111111
0000002E  24 DC 00 F5 94 6D B8 F5 09 A2 19 8C 77 ED D9 38  11111111 11111111 11111111 11111111
0000002F  3A F0 6D C1 0F 24 61 35 07 5A A1 1D FD EF 1E 9D  11111111 11111111 11111111 11111111
```

如图所示即是mp3的逻辑帧：分为header、side information、main data。header存储了采样率和声道信息，side information存储了主数据的编码信息以及方式，main data则存储了音频信息。

为了进一步压缩体积，mp3文件普遍采用了Huffman 编码。作为解码的重要一步，如何理解运用Huffman来进行相关解码操作也十分重要。

解码完成之后我们就可以获得计算机可以直接播放的源文件PCM。

本次解码的主代码来源于libmad（其是一款开源的mp3解码库，被广泛运用与各个播放器领域 <http://www.underbit.com/products/mad/>）。这个库拥有良好的mp3解码性质譬如：24bit PCM输出，百分百整数运算，基于ISO/IEC标准。我们此次作业基于这个库做了移植

与整合。（由于libmad本身学习成本较高，但是有着良好的学习受益。这里推荐两个中文文档方便学弟学习：<http://blog.chinaunix.net/uid-29068482-id-4130006.html> <http://wenku.baidu.com/view/e9420e6d9b6648d7c1c7469e.html>）

代码移植与整合的一个难点在于libmad本身基于C标准库，这个在xv6下是缺失的，因此需要去掉一些代码并重新思考实现方式。

譬如Huffman树的读取，以前的实现方式是基于文件读取。而由于相关函数缺失我们采取了直接植入源代码的方式：

```
HUFFBITS dmask = (HUFFBITS)1 << (sizeof(HUFFBITS)*8-1);

unsigned char h1[7][2] = {{ 0x2, 0x1 }, { 0x0, 0x0 }, { 0x2, 0x1 }, { 0x0, 0x10 }, { 0x2, 0x1 }, { 0x0, 0x1 }, { 0x0, 0x11
};
unsigned char h2[17][2] = {{ 0x2, 0x1 }, { 0x0, 0x0 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x10 }, { 0x0, 0x1 }, { 0x2, 0x1
}, { 0x0, 0x11 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x20 }, { 0x0, 0x21 }, { 0x2, 0x1 }, { 0x0, 0x12 }, { 0x2, 0x1 }, { 0x0
}, { 0x0, 0x22 }};
unsigned char h3[17][2] = {{ 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x0 }, { 0x0, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x11 }, { 0x2, 0x1
}, { 0x0, 0x10 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x20 }, { 0x0, 0x21 }, { 0x2, 0x1 }, { 0x0, 0x12 }, { 0x2, 0x1 }, { 0x0
}, { 0x0, 0x22 }};
unsigned char h5[31][2] = {{ 0x2, 0x1 }, { 0x0, 0x0 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x10 }, { 0x0, 0x1 }, { 0x2, 0x1
}, { 0x0, 0x11 }, { 0x8, 0x1 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x20 }, { 0x0, 0x2 }, { 0x2, 0x1 }, { 0x0, 0x21 }, { 0x0
}, { 0x8, 0x1 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x22 }, { 0x0, 0x30 }, { 0x2, 0x1 }, { 0x0, 0x3 }, { 0x0, 0x13 }, { 0
}, { 0x0, 0x31 }, { 0x2, 0x1 }, { 0x0, 0x32 }, { 0x2, 0x1 }, { 0x0, 0x23 }, { 0x0, 0x33 }};
unsigned char h6[31][2] = {{ 0x6, 0x1 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x0 }, { 0x0, 0x10 }, { 0x0, 0x11 }, { 0x6, 0x1
}, { 0x2, 0x1 }, { 0x0, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x20 }, { 0x0, 0x21 }, { 0x6, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x12 }, { 0x2
}, { 0x0, 0x2 }, { 0x0, 0x22 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x31 }, { 0x0, 0x13 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0
}, { 0x30 }, { 0x0, 0x32 }, { 0x2, 0x1 }, { 0x0, 0x23 }, { 0x2, 0x1 }, { 0x0, 0x3 }, { 0x0, 0x33 }};
unsigned char h7[71][2] = {{ 0x2, 0x1 }, { 0x0, 0x0 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x10 }, { 0x0, 0x1 }, { 0x8, 0x1
}, { 0x2, 0x1 }, { 0x0, 0x11 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x20 }, { 0x0, 0x2 }, { 0x0, 0x21 }, { 0x12, 0x1 }, { 0x6
}, { 0x2, 0x1 }, { 0x0, 0x12 }, { 0x2, 0x1 }, { 0x0, 0x22 }, { 0x0, 0x30 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x31 }, { 0
}, { 0x13 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x3 }, { 0x0, 0x32 }, { 0x2, 0x1 }, { 0x0, 0x23 }, { 0x0, 0x4 }, { 0xa, 0x1
}, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x40 }, { 0x0, 0x41 }, { 0x2, 0x1 }, { 0x0, 0x14 }, { 0x2, 0x1 }, { 0x0, 0x42 }, { 0x0
}, { 0xc, 0x1 }, { 0x6, 0x1 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x33 }, { 0x0, 0x50 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0
}, { 0x2, 0x1 }, { 0x0, 0x34 }, { 0x0, 0x5 }, { 0x0, 0x51 }, { 0x6, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x15 }, { 0x2, 0x1 }, { 0x0
}, { 0x0, 0x25 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x44 }, { 0x0, 0x35 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x53 }, { 0x0
}, { 0x54 }, { 0x2, 0x1 }, { 0x0, 0x45 }, { 0x0, 0x55 }};
unsigned char h8[71][2] = {{ 0x6, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x0 }, { 0x2, 0x1 }, { 0x0, 0x10 }, { 0x0, 0x1 }, { 0x2, 0x1
}, { 0x0, 0x11 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x21 }, { 0x0, 0x12 }, { 0xe, 0x1 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0
}, { 0x0, 0x2 }, { 0x2, 0x1 }, { 0x0, 0x22 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x30 }, { 0x0, 0x3 }, { 0x2, 0x1 }, { 0
}, { 0x31 }, { 0x0, 0x13 }, { 0xe, 0x1 }, { 0x8, 0x1 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x32 }, { 0x0, 0x23 }, { 0x2, 0x1
}, { 0x0, 0x40 }, { 0x0, 0x4 }, { 0x2, 0x1 }, { 0x0, 0x41 }, { 0x2, 0x1 }, { 0x0, 0x14 }, { 0x0, 0x42 }, { 0xc, 0x1 }, { 0x6
}, { 0x2, 0x1 }, { 0x0, 0x24 }, { 0x2, 0x1 }, { 0x0, 0x33 }, { 0x0, 0x50 }, { 0x4, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x43 }, { 0
}, { 0x34 }, { 0x0, 0x51 }, { 0x6, 0x1 }, { 0x2, 0x1 }, { 0x0, 0x15 }, { 0x2, 0x1 }, { 0x0, 0x5 }, { 0x0, 0x52 }, { 0x6, 0x1
}, { 0x2, 0x1 }, { 0x0, 0x25 }, { 0x2, 0x1 }, { 0x0, 0x44 }, { 0x0, 0x35 }, { 0x2, 0x1 }, { 0x0, 0x53 }, { 0x2, 0x1 }, { 0x0
}, { 0x54 }, { 0x2, 0x1 }, { 0x0, 0x55 }};
```

诸如此类有很多小技巧，但是从整体上破坏了代码结构的美感。因此需要与其他文件小组成员进一步深化合作，完成改进。

解码部分的详尽代码请看 play.c 里面的 decode 函数以及 decodemp3.c 相关文件。

---

## mp3播放

播放更多的涉及到的是一些进程通信和系统调用。由于我们本次并没有完美实现mp3流畅播放（我们是解码完成之后直接播放PCM文件，因为解码的过程实在是太慢），因此这里剖析一下上一届的一些代码流程。

首先我们需要一些系统调用用以实现播放：

- `setSampleRate`：设置声卡采样率。
- `writeaudio`：传递读取的文件到内核。
- `setVolume`：设置声音大小。
- `pause`：通知内核暂停播放。
- `wavdecode`：开始缓存用户读取的音频数据。
- `beginDecode`：开始进行mp3解码。
- `waitForDecode`：等待上一个mp3解码完成。
- `endDecode`：设置mp3解码状态为已完成。
- `getCoreBuf`：从内核态获取音频数据用于解码。

具体的播放流程就是首先 `play` 函数不断读取数据调用 `beginDecode`函数唤醒 `decode` 函数用以解码。根据 `decode` 函数状态决定阻塞与唤醒。`decode` 函数则从数据共享区里面拿出解压数据，予以解码并将数据放入音频缓冲区。最后 `play` 函数对音频缓冲区里面的文件予以播放。

我们的代码存在一些差错导致解码和播放时会出现内存越界和缺页的错误。这个错误我们会尽快予以更正。最后实现流畅的播放。

---

## 经验与教训

在这次不完美的作业中，我们在mp3播放里遇到了很多困难，同时也收获了很多经验。这里做简要陈述：

### Debug

初期，主要的调试方法为gdb。在xv6根目录下，打开两个命令行窗口，A窗口运行make qemu-gdb，B窗口运行gdb kernel，此时在B窗口中可以通过输入b[reak] func加入断点，还可以输入c[ontinue]执行下一步，p[rint] variable输出变量值。尽管gdb在linux环境下，是一个比较完善的调试工具，然而，在对xv6进行调试时，嵌套层数很多，而且问题往往在汇编层面才显现出来，因而调试难度很大。

因而后来主要采用输出特定信息的方式来确定问题所在。在需要调试的程序中加入若干printf语句输出标志量。这是一种比较原始的方式，效率很低，每次修改输出位置或者输出信息，都需要重新编译、运行。所以后来在调试解码程序的时候，便将代码移植到VS环境下调试，当调试完成，再移植到xv6中。

### Memory

在把调试好的解码程序移植到xv6环境下后，在运行时出现如下问题：

```
pid 3 play: trap 14 err 4 on cpu 1
```

在traps.h中对trap 14的定义为

```
#define T_PGFLT 14 // page fault
```

所以认为问题是CPU访问内存时，发生了缺页错误，导致程序中断。

一度怀疑是xv6的Lazy Page Allocation机制的锅，因为解码程序中创建了很多大数组，可能出现了内存不够的问题。不过很快就打消了这种顾虑，在VS下运行解码程序，所占内存一直稳定在2M以下。现在一直卡在这个问题上，没法取得新的进展，还望学弟学妹们能实现吾之未竟大业！

### Lock

为了进一步解决内存问题，我们当时认为是由于核心数据没有上锁导致双CPU同时修改核心数据进而引发错误。仔细研读xv6 Lock 章节的代码，运用 spinlock 在解码前后都予以上锁。问题并没有得以解决。希望学弟学妹们不要再次尝试用锁解决问题

### People

---

本次大作业其实也涉及到很多任务分配不够清晰，日程规划不够完善的缺陷。而xv6的代码缺乏合理有效的调试手段也一直阻碍我们的进度有明显提升。

希望以后的小组能够吸收我们的经验教训。完成好xv6的整合！

## 人员分工

肖剑楠	声卡资料查阅，相关库函数实现，调试
董亮	播放代码整合，调试
卢凌铜	解码代码整合，调试
杨志强	调试