# The μClinux Audio Player Document

Version 0.1

Meihui Fan
范 美 辉
<mhfan@ustc.edu>



2004年12月21日

*http://www.hhcn.com

## Thanks

特别感谢张翼在 μCAP 开发过程中的指导和帮助， 感谢王刚在 μCAP 与 GUI 整合过程中的协同合作。 此外，黄光华(yehuang)原来的实现给了我很多启发。

## Abstract

*Keywords:* **Embedded, μClinux, Audio Player, Decoder, MP3, WMA, WAV, OGG.**

# 目录

# 1 Introduction

μCAP 是 μClinux Audio Player 的首字母缩写。顾名思义，这是特别为我们华恒公司的 Forthgoer 项目[1]而设计开发的运行在 μClinux 操作系统下的嵌入式软件音频播放器[2]。

## 1.1 Prerequires

μCAP 的软硬件要求：(以下所列一般都是可以但不必须的要求，请自己尝试其它的软硬件条件。)

**CPU:** Motorola ColdFile 5249.

**Memory:** SRAM ≥ 64 KB; SDRAM ≥ 16 MB.

**Board:** HHCF5249 develop board.

**OS platform:** μClinux-dist-040513.

**Toolchain:** m68k-elf-tools20030314.

**Libraries:** μC-libc.

## 1.2 Features

μCAP 的设计开发目标是全面降低音频解码回放过程中的硬件功耗，同时兼顾用户操作的 响应。它的主要特性包括：

- ✓ CPU 占用率低。经不严格的粗略测试， μCAP 在对各种音频格式进行正常的连 续解码回放过程中，其 CPU(MCF5249: 工作频率 140 MHz) 占用率不超过 %25。
- ✓ 磁盘访问率低。充分利用主存资源做为解码器的磁盘缓存，从而有效降低磁盘访问率。
- ✓ 快速的用户操作响应。从播放的起始到连续的快进、快退等都力求让用户感觉不到响应延迟。
- ✓ 支持 MP3、WMA、WAV、OGG(尚未加入) 等音频格式的解码回放。
- ✓ 正常处理音频文件中的标签信息(中文简繁体和英文等)。
- ✓ 支持 A-B 段记忆回放功能(尚未完全实现)。

[1] Forthgoer project: 华恒公司的硬盘 MP3 产品方案。
[2] μCAP 开始称为‘mp3play’，最初由黄光华(yehuang)实现。

# 2 Design Scheme

μCAP 的主要设计思想是开辟空闲的 SDRAM 内存空间作为音频编码文件的磁盘缓存，以 尽可能地减少磁盘访问，从而达到节能省电的最终目的。

由于缓存空间较大，为了使最终用户感觉不到缓存(延时)的存在，我们需要让播放器解码 ‘动作’和磁盘缓存‘动作’分别是两个并发执行的进程。所以必须小心定义这两个同步 进程的通讯机制。

## 2.1 Program Flow

现把 μCAP 的主要程序流程(程序流程图见图**??**)简要描述如下：

μCAP 正常启动(由 GUI fork 出来，或者直接命令行启动)之后首先按照既定的播放模式 初始化播放列表，然后 fork 出磁盘缓冲进程(hdload)并同步等待其初始化完成，接着打开 并初始化设置声卡设备(/dev/dsp)，最后进入回放循环。

在回放循环中，依据播放列表中当前要播放的文件的音频格式调用相应的解码器。在解码 器初始化完成之后进入解码循环之前处理并输出歌曲的标签信息(标题和作者)在解码 循环中(具体来说在每个解码单元的解码完成之后即将输出之前)处理用户指令(改变 播放模式，快进快退等)和输出播放进度(播放时间和播放比例)。

## 2.2 Buffering

磁盘缓冲是 μCAP 实现的关键。

### 2.2.1 Size & Allocation

为了减小启动的延时，我们让操作系统预留出8 MB 的 SDRAM 空间做为公共的缓存和 共享内存使用。当然，这不是必须的；稍作修改就可以利用malloc 来分配缓存空间。

这些缓存空间等分成固定大小(16 KB)的缓存单位，并以双向链表的数据结构来组织。选 择使用双向链表完全是为了访问的快速和实现的方便，实际上对于连续的缓存空间来说这 里应该还有改进的空间。

### 2.2.2 Strategy

为了尽可能地减少磁盘访问，我们采用全缓冲的策略。具体来说，就是尽量填满缓存空间 ；并且在正常的顺序缓冲的情况下，每次缓冲一开始就重新缓存全部空间。当然在细节的 实现上必须考虑一些异常的特殊情况，包括：

- 大文件的缓冲策略。
- 文件的缓冲区覆盖问题。
- 播放时，快进快退操作对缓冲的影响。

### 2.3 Inter Processes Communication

系统中与音频解码回放相关的进程主要有三个：GUI 进程，回放进程和硬盘缓冲进程；它 们之间主要是协同关系。在实现中我们主要用信号量来同步进程，用共享内存来做通信。 因为用信号量做同步简单而有效，而在 μC 系统中，各进程共用同一个地址空间， 用共享内存做通信最为方便。

## 3 Audio Decoder

### 3.1 MP3 Decoder

关于 MP3 解码器接口的细节请参考：**??**、**??**、**??**。

**mp3d_ptr** ：指向 MP3 解码器结构的指针。

**init_mp3d()** ：初始化 MP3 解码器。

**mp3_info()** ：计算 MP3 的码流信息。

**mp3_decode_loop()** ：MP3 解码循环。

**mp3d_->fun_ptr_MP3D_decode_frame** ：解码一个缓冲区中的 编码帧。

**mp3d_swap_buffers(char**pbuf, int*plen)** ：切换 MP3 的解 码缓冲区。
注意：这个函数的接口调用存在 Bug(pbuf 的类型不一致，所以它 不能正确传递缓冲区的地址)。必须在实现的时候把缓冲区的地址直接赋给全局变量——‘mp3d_ptr->tbl_MP3D_in_buf_ptr’。

### 3.2 WMA Decoder

关于 WMA 解码器接口的细节请参考：**??**、**??**、**??**。

**init_wmad()** ：初始化 WMA 解码器。

**wma_info()** ：处理 WMA 的标签信息，并做编码转换。

**wma_decode_loop()** ：WMA 解码循环。

**_Linux_WMAD_FileDecodeDataSubFrames** ：解码一个缓冲区。

**_Linux_WMAD_Get_New_Data(unsigned char **pbuf, int *plen, int *poff, int *need_seek)**
：获取新的解码缓冲区。
注意：这几个参数在内存(SRAM)中的位置是固定的！

### 3.3 WAV Decoder

**init_wavd()** ：初始化 WAV 解码器。

**wav_info()** ：处理 WAV 文件的相关信息。

**wav_GetFrequency()** ：取得 WAV 编码的频率。

**wav_GetTotalTime()** ：取得 WAV 编码总播放时间。

**wav_GetDecodedTime()** ：取得当前解码耗费时间。

**wav_AdpcmDecode()** ：解码一个缓冲区。

### 3.4 OGG Decoder

有待实现 ...

## 4 C Functions Prototype

C 语言函数原型和有关实现细节的一些说明。

Common Definitions and Data Structures

```
/*****************************************************************
 * $ID: common.h       Fri, 30 Jul 2004 15:52:35 +0800  mhfan $ *
 *                                                               *
 * Description:                                                  *
 *                                                               *
 * Maintainer:  范美辉(Meihui Fan)  <mhfan@ustc.edu>            *
 *                                                               *
 * CopyRight (c)  2004  HHTech                                   *
 *   www.hhcn.com, www.hhcn.org                                  *
 *   All rights reserved.                                        *
 *                                                               *
 * This file is free software;                                   *
 *   you are free to modify and/or redistribute it               *
 *   under the terms of the GNU General Public Licence (GPL).    *
 *                                                               *
 * Last modified: Wed, 01 Dec 2004 17:31:59 +0800      by mhfan #
 *****************************************************************/
#ifndef COMMON_H
#define COMMON_H
typedef struct buf_info {
    int     frames;                 // how many frames in this buffer
    struct file_info* fi;           // occupied by the file
    struct buf_info *prev, *next;   // double linked list
    void*   base;                   // real base address
```

```
25  }   BufInfo;
26
27  typedef struct file_info {          // too big a struct
28      char*  path;                    // path to the file
29      AudioFmt fmt;                   // audio format
30      off_t  size, offset, loff;      // file size, overwrited & loadded offset
31      struct { char    *title, *artist; } id3tag;     // only for mp3
32      struct { BufInfo *begin, *end, *cur; } bufs;    // occupying buffers
33      struct file_info *prev,  *next; // double linked list
34  }   FileInfo;
35
36  typedef struct play_list {
37      int      sum;                   // all files sumary
38      PlayRule rule;                  // play rule
39      FileInfo *fiarr;                // file-info array
40      FileInfo *lcur, *pcur;          // loading and playing cursor.
41      off_t    poff, pa, pb;          // playing offset and memory point A & B
42  }   PlayList;
43
44  #endif//COMMON_H
45  /******************** End Of File: common.h ********************/
```

## 4.1 hdload

音频文件的磁盘缓冲进程。

### 4.1.1 Data Structure

**BufInfo** ：双向链接的缓冲区列表。

Refer to A。It is self-documented.

### 4.1.2 Application Interface

**hdload()** ：从硬盘文件中填充缓冲区。

### 4.1.3 Program Flow

程序流程主要是：

1. ...

## 4.2 μCAP

音频解码和回放进程。

#### 4.2.1 Data Structure

**PlayList** ：播放列表(not a real list)。

**FileInfo** ：双向链接的播放文件信息列表。

Refer to A。They are self-documented.

#### 4.2.2 Application Interface

**init_playlist()** ：根据播放规则初始化播放列表。

**reset_playlist()** ：在遍历播放列表之后，根据播放模式重排播 放列表。

**proc_cmd()** ：在解码循环中处理用户提交的播放命令。

**preload()** ：在切换解码缓冲区和切换播放歌曲时，缓冲文件。

**play_progress()** ：计算播放进度。

#### 4.2.3 Program Flow

程序流程主要是：

1. ...

## 5 GUI ⇌ UCAP Shared Memory

### 5.1 Proposed Shared Memory Structure

GUI ⇌ μCAP Shared Memory Definition

```
/*****************************************************************
 * $ID: shmem.h        Mon, 23 Aug 2004 11:19:31 +0800  mhfan $ *
 *                                                               *
 * Description:                                                  *
 *                                                               *
 * Maintainer:  范美辉(Meihui Fan)  <mhfan@ustc.edu>            *
 *                                                               *
 * CopyRight (c)  2004  HHTech                                   *
 *   www.hhcn.com, www.hhcn.org                                  *
 *   All rights reserved.                                        *
 *                                                               *
 * This file is free software;                                   *
 *   you are free to modify and/or redistribute it               *
 *   under the terms of the GNU General Public Licence (GPL).    *
```

```
 *                                                                  *
 * Last modified: Wed, 01 Dec 2004 17:56:56 +0800      by mhfan #
 ********************************************************************/
#ifndef SHMEM_H
#define SHMEM_H
typedef struct __attribute__((packed)) shm_data {
    /* following are information used by communication. GUI <==> UCAP */
    int sign;                       // sign for GUI-UCAP valid
    int inform;                     // inform UCAP to receive command
    int command;                    // command word

    /* following are the playing information.          GUI <=== UCAP */
    int feedback;                   // communication feedback by UCAP.
    int playidx;                    // current playing file index
    int duration;                   // duration time (seconds)
    int elapsed;                    // elapsed  time (seconds)

    int ratio;                      // percent ratio
    int rule;                       // play rule
    int freq;                       // (KHz)
    int bitrate;                    // (Kbps)
    char title [MAX_TAGINFO_SIZE];  // Tag infor
    char artist[MAX_TAGINFO_SIZE];  // Tag infor

    /* following are the playlist information.         UCAP <=== GUI */
    int first;                      // first file index of the playlist
    int last;                       //  last file index of the playlist
    int selidx;                     // selected file index
    int fsum;                       // files sumary number

    char* *path;                    // pointer to the files array
}   ShmData;

#endif//SHMEM_H
/********************* End Of File: shmem.h *********************/
```

## 5.2 Communication Mechanism

### 5.2.1 GUI ⇀ μCAP

### 5.2.2 GUI ↽ μCAP

```
/*
 *  几点说明:
 *
 * 1. 共享内存采用结构体定义以增加可维护性和可移植性。
 *
 * 2. 数值信息尽量定义为 int 以提高存储效率。
```

```
*
* 3. 数值信息的有效性可以用正负值来标志
* （比如：规定正值有效——比特率的负值表示‘VBR’）。
*
* 4. 播放规则不显式区分所有文件、某一目录、单个文件等，
* 只分为顺序播放、随机播放、循环播放以及三者的组合情况
* （如：顺序循环、随机循环）；
* 要求 GUI 每次都给出当前需要播放的文件列表的首尾指针
* （建议采用前闭后开区间的方式）。
*
* 5. 通讯协议：（暂可按照原先的定义）。
* （个人觉得采取管道方式实现更简单一些，
* 同时也可以避免过多的休眠状态）。
*
* 6. 通讯过程：
* (1). GUI ==> UCAP:
*      a. GUI fork 启动 UCAP，置：      shm->sign  = 1
* 并置：      shm->comm  = COMM_INVALID，
* 然后等待 UCAP 的反馈：      shm->comm ?= COMM_READY;
* 如果反馈成功则置：      shm->comm  = COMM_INVALID;
*
*      b. 用户出发新的命令时，GUI 置   shm->comm  = COMM_COMMAND，
* 然后等待 UCAP 的反馈：      shm->comm ?= COMM_SUCCESS;
* 如果 shm->comm == COMM_FAILD，则重发命令（重复 b 步骤）；
* 如果成功则置：      shm->comm  = COMM_INVALID;
*
*      c. 注意：
* 每次通讯成功，GUI 必须置：   shm->comm  = COMM_INVALID;
*
* (2). UCAP ==> GUI:
*      a. UCAP 由 GUI fork 启动后置    shm->comm  = COMM_READY;
*
*      b. UCAP 每帧解码前检查：      shm->comm ?= COMM_COMMAND;
* 若是则读取命令字并做相应处理，
* 同时置：      shm->comm  = COMM_SUCCESS，
* 若命令无效，置：      shm->comm  = COMM_FAILD;
*
*      c. 注意：
* 每次UCAP要反馈时，必须检查：shm->comm ?= COMM_INVALID;
*
*/
```

# 6 Problems & BUGs

留存的问题、Bug，以及计划实现的特性：

- A-B 段记忆回放功能的实现有待测试。
- 进程间同步通信过程仍有改进的空间。
- …

# 7 Experience

以下是我个人关于 μCAP 开发过程中的一些经验总结。

## 7.1 Compilation & Linking

鉴于 μC 系统内存管理的薄弱，它不适于引入动态连接和加载的机制。所以基于 μC 的应用开发通常利用静态编译。

一个静态连接库文件实际上只是编译对象文件 (object) 的汇集。连接器在连接的时候， 从左到右地扫描每个对象文件，依此分析和重定位其中的符号，并收集其中的未定义符号 ；在这个过程中每遇到一个静态连接库文件就查询和重定位已收集的未定义符号。重定位 后符号的位置与文件之间的顺序有极大的关系，所以对于短指针类型的符号在连接的时候 应该把它们所在的文件尽可能地靠近。

## 7.2 Some Macros

**GUI_CONTROL** ：打开与 GUI 进程同步通信的机制，接受 GUI 的播放控制。

**CLI_CONTROL** ：接受命令行(标准输入)的控制序列，并在标准输出给出播放状 态的信息；可以与 “GUI_CONTROL”宏并存。

**DEBUG** ：在标准错误上输出调试信息。

**DUMP_OUTPUTS** ：将音频解码输出存储为文件。

**Others** ：Refer to ‘include/common.h’.

## 7.3 Development Details

- MCF5249 is a *big-endian* machine.
- 嵌入式平台 IDE 接口的磁盘访问速度低， 故存储器映射文件并不能显著提高效率。

- μC-libc 的信号通讯机制的实现似乎不完善， setsigjmp/siglongjmp 不能正常地自动恢复信号屏蔽/阻塞。
- 信号相当于软中断，所以要注意在‘中断处理程序’中‘清中断’(即在信号处 理程序中阻塞信号)。
- ……

# 参考资料


- “*MP3 Extensions*”, Alex Beregszaszi, November 28, 2003. (http://home.pcisys.net/ melanson/codecs/mp3extensions.txt)
  *provide the informal explaination of MP3 Xing VBR header, LAME header, etc.*
- “*MP3 Decoder on* μ*Clinux Release Notes*”, V 0.4, Motorola India Electronics Pvt. Limited, 2004.
- “*MP3 Decoder on* μ*Clinux Demo User Manual*”, V 0.4, Motorola India Electronics Pvt. Limited, 2004.
- “*Interface Definition Document for MP3 Decoder on* μ*Clinux*”, V 1.3, Motorola India Electronics Pvt. Limited, 2004.
- “*WMA Decoder on* μ*Clinux Release Notes*”, V 0.4, Motorola India Electronics Pvt. Limited, 2004.
- “*WMA Decoder on* μ*Clinux Demo User Manual*”, V 0.4, Motorola India Electronics Pvt. Limited, 2004.
- “*Interface Definition Document for WMA Decoder on* μ*Clinux*”, V 1.3, Motorola India Electronics Pvt. Limited, 2004.
- “*Advanced Programming in the UNIX Environment*”, W. Richard Stevens, 1992, Addison Welsley Publishing Company.

# A Part of C sources

Implememtion of 'hdload'

```
/*****************************************************************
 * $ID: hdload.c       Fri, 30 Jul 2004 13:05:24 +0800  mhfan $ *
 *                                                               *
 * Description:                                                  *
 *                                                               *
 * Maintainer:  范美辉(Meihui Fan)  <mhfan@ustc.edu>             *
 *                                                               *
 * CopyRight (c)  2004  HHTech                                   *
 *   www.hhcn.com, www.hhcn.org                                  *
 *   All rights reserved.                                        *
 *                                                               *
 * This file is free software;                                   *
 *   you are free to modify and/or redistribute it               *
 *   under the terms of the GNU General Public Licence (GPL).    *
 *                                                               *
 * Last modified: Wed, 01 Dec 2004 16:37:45 +0800      by mhfan #
 *****************************************************************/

#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <sys/types.h>
#include <sys/resource.h>

#include "id3tag.h"
#include "common.h"

#ifndef DEBUG_HDLOAD
#undef  dtrace
#undef  dprint
#define dtrace
#define dprint(...)
#endif//DEBUG_HDLOAD

int fd=-1;
#ifdef  CFG_PM
int pmfd=-1;
#endif//CFG_PM
int loaded_buf=0;
sigset_t sigmask, sigset;
static sigjmp_buf jmpbuf;
BufInfo bufs_arr[BUF_NUM];
PlayList* playlist=(PlayList*)(GUI_SHM_BASE - sizeof (PlayList));

char* tagstr(struct id3_tag const *tag, char const *id)
```

```
46  {   // get the tag string from ID.
47      struct id3_frame const *frame;
48      union id3_field const *field;
49      unsigned long const *ucs4;
50
51      frame = id3_tag_findframe(tag, id, 0);
52      if (frame == 0) return NULL;
53      field = id3_frame_field(frame, 1);
54      ucs4  = id3_field_getstrings(field, 0);
55      if (!strcmp(id, ID3_FRAME_GENRE))
56          ucs4 = id3_genre_name(ucs4);
57      return id3_ucs4_latin1duplicate(ucs4);
58  }
59
60  void init_bufs()
61  {
62      int i;  void* base=(void*)BUF_BASE;
63
64      for (i=0; i<BUF_NUM; ++i) {
65          bufs_arr[i].fi = NULL;
66          bufs_arr[i].base = base;
67          base += BUF_MAX_SIZE;
68          bufs_arr[i].next = &bufs_arr[i+1];
69          bufs_arr[i].prev = &bufs_arr[i-1];
70      }   dtrace;
71      bufs_arr[BUF_NUM-1].next = &bufs_arr[0];
72      bufs_arr[0].prev = &bufs_arr[BUF_NUM-1];
73  }
74
75  static inline FileInfo* next_play(FileInfo* fi)
76  {
77      FileInfo* tfi=fi;
78      for (fi=fi->next; fi != tfi; fi=fi->next) {
79          if ((loaded_buf += fi->loff / BUF_MAX_SIZE) >= BUF_NUM) break;
80          if (fi->loff < fi->size || fi->offset > 0) return fi;
81      }                                                  return NULL;
82  }
83
84  void hdload()
85  {
86      FileInfo *fi, *tfi;
87
88  #ifdef  CFG_PM
89      ioctl(pmfd, PM_SET_CPU_FREQ, 140);
90      ioctl(pmfd, PM_LOCK_CPU_FREQ);
91  #endif//CFG_PM
92  NEXT:
93      fi = playlist->lcur;
94      if (fd != -1) close(fd);
```

```
95      dprint("\033[1mload␣file:␣%s\n", fi->path);
96      if ((fd = open(fi->path, O_RDONLY)) < 0) {
97          dprint("Can't␣open␣'%s'(fi=%p)!\n", fi->path, fi);
98          fi->size = 0;                                       goto RTN;
99      }
100
101     if (fi->loff == 0) { fi->offset = 0;
102         fi->bufs.begin = fi->bufs.end = (fi->prev->bufs.end ?
103                          fi->prev->bufs.end : &bufs_arr[loaded_buf]);
104         if (fi->size == INVALID_SIZE) {
105             if (fi->fmt == AUDIO_MP3) {
106                 struct id3_tag* id3tag =
107                        id3_file_tag(id3_file_fdopen(fd, 0));
108                 fi->id3tag.title  = tagstr(id3tag, ID3_FRAME_TITLE);
109                 fi->id3tag.artist = tagstr(id3tag, ID3_FRAME_ARTIST);
110             }   fi->size = lseek(fd, 0, SEEK_END);
111         }
112     } else if (fi->offset > 0) {
113         dprint("%s:␣loff=%lx,␣offset=%lx\n", fi->path, fi->loff, fi->offset);
114         lseek(fd,  fi->offset - BUF_MAX_SIZE, SEEK_SET);
115         while (loaded_buf++ < BUF_NUM) {
116             if ((tfi = fi->bufs.begin->prev->fi)) {
117                 sigprocmask(SIG_BLOCK,  &sigmask, NULL);//   block SIG_LOAD
118                 tfi->bufs.end->fi = NULL;
119                 tfi->bufs.end = tfi->bufs.end->prev;
120                 tfi->loff -= BUF_MAX_SIZE;
121                 sigprocmask(SIG_SETMASK, &sigset, NULL);// unblock SIG_LOAD
122             }
123
124             read (fd, fi->bufs.begin->prev->base, BUF_MAX_SIZE);
125
126             sigprocmask(SIG_BLOCK,  &sigmask, NULL);    //   block SIG_LOAD
127             fi->bufs.begin = fi->bufs.begin->prev;
128             fi->bufs.begin->fi = fi;
129             fi->offset -= BUF_MAX_SIZE;
130             sigprocmask(SIG_SETMASK, &sigset, NULL);    // unblock SIG_LOAD
131
132             if (fi->offset == 0) break;
133             lseek(fd, -(BUF_MAX_SIZE<<1), SEEK_CUR);
134         }                                                   goto RTN;
135     }       lseek(fd, fi->loff, SEEK_SET);
136
137 dtrace;
138     while (loaded_buf++ < BUF_NUM) {
139         if ((tfi = fi->bufs.end->fi)) {
140             sigprocmask(SIG_BLOCK,  &sigmask, NULL);    //   block SIG_LOAD
141             tfi->bufs.begin->fi = NULL;
142             tfi->bufs.begin = tfi->bufs.begin->next;
143             tfi->offset += BUF_MAX_SIZE;
```

```
144             sigprocmask(SIG_SETMASK, &sigset, NULL);    // unblock SIG_LOAD
145         }
146
147         if ((read(fd, fi->bufs.end->base, BUF_MAX_SIZE)) == 0) {
148             dprint("\033[1m%s:␣begin=%p,␣end=%p,␣loff=%ld\n", fi->path,
149                     fi->bufs.begin->base, fi->bufs.end->base, fi->loff);
150             if (tfi) {
151                 sigprocmask(SIG_BLOCK,  &sigmask, NULL);//   block SIG_LOAD
152                 tfi->bufs.begin = tfi->bufs.begin->prev;
153                 tfi->bufs.begin->fi = tfi;
154                 tfi->offset -= BUF_MAX_SIZE;
155                 sigprocmask(SIG_SETMASK, &sigset, NULL);// unblock SIG_LOAD
156             }
157             if ((playlist->lcur=next_play(fi)))         goto NEXT;
158             else                                        goto RTN;
159         }
160
161         sigprocmask(SIG_BLOCK,  &sigmask, NULL);        //   block SIG_LOAD
162         fi->bufs.end->fi = fi;
163         fi->loff += BUF_MAX_SIZE;
164         fi->bufs.end = fi->bufs.end->next;
165         sigprocmask(SIG_SETMASK, &sigset, NULL);        // unblock SIG_LOAD
166     }
167     dprint("\033[1m%s:␣begin=%p,␣end=%p,␣loff=%ld\n", fi->path,
168             fi->bufs.begin->base, fi->bufs.end->base, fi->loff);
169 RTN:
170     loaded_buf = 0;
171     playlist->lcur = NULL;
172 #ifdef  CFG_PM
173     ioctl(pmfd, PM_UNLOCK_CPU_FREQ);
174     ioctl(pmfd, PM_SET_CPU_FREQ, 56);
175 #endif//CFG_PM
176 }
177
178 void sighdl_load(int signum)
179 {
180 dtrace;
181 #ifdef  CFG_PM
182     ioctl(pmfd, PM_UNLOCK_CPU_FREQ);
183 #endif//CFG_PM
184     siglongjmp(jmpbuf, signum);
185 }
186
187 void sighdl_term(int signum)
188 {
189 dtrace;
190 #ifdef  CFG_PM
191     ioctl(pmfd, PM_UNLOCK_CPU_FREQ);
192 #endif//CFG_PM
```

```
193     exit(0);
194 }
195
196 int main(void)
197 {   // do initialization, then pause to wait for signal.
198
199     setpriority(PRIO_PROCESS, 0, -19);      init_bufs();
200
201 #ifdef  CFG_PM
202     if((pmfd = open(PM_DEVICE, O_RDONLY)) < 0)
203         dprint("Can't␣open␣Power␣Maneger␣device!\n");
204 #endif//CFG_PM
205
206     { struct sigaction sar;
207     sar.sa_flags = 0;
208     sigemptyset(&sar.sa_mask);
209     sar.sa_handler = sighdl_load;
210     if (sigaction(SIG_LOAD, &sar, NULL) < 0) EXIT(ERR_SIG_INIT);
211
212     sar.sa_handler = sighdl_term;
213     if (sigaction(SIGTERM,  &sar, NULL) < 0) EXIT(ERR_SIG_INIT);
214     }
215
216     sigemptyset(&sigmask);
217     sigaddset(&sigmask, SIG_LOAD);
218     sigprocmask(0, NULL, &sigset);  // save the signal mask set
219
220     if (kill(getppid(), SIG_READY))          EXIT(ERR_SIG_SEND);
221     else dprint("###␣hdload␣has␣already␣initialized.\n");
222
223     if (sigsetjmp(jmpbuf, 1)) {
224         sigprocmask(SIG_SETMASK, &sigset, NULL);      hdload();
225     }
226
227     for (;;) pause();
228 }
229
230 #if 0   /* comment by mhfan */
231 int stricmp(const char* s, const char* d)
232 {   // defined by mhfan, an implementation of case-insensitive 'strcmp'.
233     while (*s!='\0' && *d!='\0' && tolower(*(s++))==tolower(*(d++))) ;
234     return *s != *d;
235 }
236 #endif /* comment by mhfan */
237
238 /******************** End Of File: hdload.c ********************/
```

Implememtion of 'ucap'

```
1 /*****************************************************************
```

```
2   * $ID: ucap.c          Sun, 01 Aug 2004 18:15:14 +0800  mhfan $ *
3   *                                                                *
4   * Description:                                                   *
5   *                                                                *
6   * Maintainer:  范美辉(Meihui Fan)  <mhfan@ustc.edu>             *
7   *                                                                *
8   * CopyRight (c)  2004  HHTech                                    *
9   *   www.hhcn.com, www.hhcn.org                                   *
10  *   All rights reserved.                                         *
11  *                                                                *
12  * This file is free software;                                    *
13  *   you are free to modify and/or redistribute it                *
14  *   under the terms of the GNU General Public Licence (GPL).     *
15  *                                                                *
16  * Last modified: Thu, 09 Dec 2004 10:42:38 +0800      by mhfan #
17  ******************************************************************/
18
19  #include <linux/soundcard.h>
20
21  #include "wmadec.h"
22  #include "mp3dec.h"
23  #include "common.h"
24
25  #define FRAME_SIZE                  WMA_FRAME_SIZE
26  #define SARCO_BUF_SIZE              (2*WMA_DEC_BUF_SIZE)
27  //#define       FRAME_SIZE                  MP3_FRAME_SIZE
28  //#define       SARCO_BUF_SIZE              (3*MP3_DEC_BUF_SIZE)
29  #define MAX_WAIT_LOOP               256
30
31  #define PRINT_VERSION               do { \
32          fprintf(stdout, "\033[2J\n\033[1;33mThe␣\033[31muC\033[33mLinux␣" \
33                          "\033[31mA\033[33mudio␣\033[31mP\033[33mlayer" \
34                          "\033[0m␣(\033[32mV␣\033[1m0.1\033[0m)\n" \
35                          "\033[1;34m---␣maintained␣by␣" \
36                          "\033[36mMeihui␣Fan␣\033[0;37m" \
37                          "<\033[35mmhfan@ustc.edu\033[37m>\033[0m\n\n" \
38                  ); } while (0)      /* mhfan */
39
40  /* global variable definitions */
41  int dspfd;
42  pid_t pid_hdl;
43  int play_frames;
44  ShmData* shm=(ShmData*)GUI_SHM_BASE;
45  CmdStatus cmd_status = CMD_STAT_READY;
46  PlayList* playlist=(PlayList*)(GUI_SHM_BASE - sizeof (PlayList));
47
48  tWMAFileHeader* wma_fh;
49  short buf_out_pcm[2*FRAME_SIZE];
50  long sram_ptr = SRAM_BASE;
```

```
51 #ifndef DISABLE_SARCO
52 short sarco_out_buf[SARCO_BUF_SIZE];
53 SARCO_global_struct* sarco_ptr;
54 #endif//DISABLE_SARCO
55
56 void init_playlist(int fsum, char* path[]);
57 void reset_playlist(int first, int last, PlayRule rule);
58 CmdStatus proc_cmd();
59 void preload(FileInfo* fi);
60
61 extern void mp3dec(void);
62 extern void init_mp3d(void);
63 extern int  init_wmad(void);
64 extern void wmadec(void);
65
66 static void sighdl_ready(int signum) { }
67
68 static void sighdl_term(int signum)
69 {
70     free(playlist->fiarr);            close(dspfd);
71 #ifdef  CLI_CONTROL
72     fprintf(stdout, "End␣of␣playback,␣Goodbye!\n");
73 #endif//CLI_CONTROL
74     if (kill(pid_hdl, SIGTERM)) EXIT(ERR_SIG_SEND);
75     usleep (1000);
76 }
77
78 static inline void init_dsp(int freq, int stereo, int playbits)
79 {
80     // Check if data stream is stereo, otherwise must play mono.
81     int bits = playbits==16 ? AFMT_S16_BE : AFMT_U8;
82     if (ioctl(dspfd, SNDCTL_DSP_SPEED, &freq)      < 0 || // slowly
83         ioctl(dspfd, SNDCTL_DSP_STEREO, &stereo)   < 0 ||
84         ioctl(dspfd, SNDCTL_DSP_SAMPLESIZE, &bits) < 0)
85         dprint("ERROR:␣Unable␣to␣initialize␣the␣DSP␣defice!\n");
86 }
87
88 static inline void clear_dspbuf()
89 {
90 #define SNDCTL_DSP_CLEARBUF     _SIO ('P', 11)
91     int speed = 0;
92     ioctl(dspfd, SNDCTL_DSP_SPEED, &speed);     // slowlly blocked
93     ioctl(dspfd, SNDCTL_DSP_CLEARBUF);
94     speed = 44100;
95     ioctl(dspfd, SNDCTL_DSP_SPEED, &speed);
96 }
97
98 void play_progress(/*int sec*/)
99 {
```

```
100     int sec=0, wav_GetDecodedTime(void);
101     switch (playlist->pcur->fmt) {
102     case AUDIO_MP3:
103         sec = play_frames * 418/*bytes*/ / (128/*Kbps*/ * 125); break;
104     case AUDIO_WMA:
105         sec = play_frames * WMA_FRAMES_PER_THOUSAND / 1000;     break;
106 #if 0   /* comment by mhfan */
107             play_frames *
108                 (playlist->poff - wma_fh->first_packet_offset) /
109                  wma_fh->packet_size / 1000;
110             (float)((float)wma_fh->duration *
111             (float)(playlist->poff - wma_fh->first_packet_offset) /
112             (float)(wma_fh->last_packet_offset - wma_fh->first_packet_offset +
113                     wma_fh->packet_size) / (float)1000);
114 #endif  /* comment by mhfan */
115     case AUDIO_WAV:
116         sec = wav_GetDecodedTime();
117     case AUDIO_OGG:             case AUDIO_UNK:
118     }
119     // considered the PCM-DSP buffer here
120 #ifdef  GUI_CONTROL
121     { int ratio = playlist->poff * 100 / playlist->pcur->size;
122     shm->elapsed = sec;         shm->ratio = ratio;
123     }
124 #endif//GUI_CONTROL
125 #ifdef  CLI_CONTROL
126     {   int hr=0, min=0, i;
127     if (sec > 3599) { hr = sec / 3600;  sec %= 3600; }
128     if (sec > 59)   { min= sec / 60;    sec %= 60;   }
129     ratio = ratio * 7 / 10;
130     fprintf(stdout, "␣\033[32m");
131     for (i=0; i < ratio; ++i) fprintf(stdout, "#");
132     for (; i < 70; ++i)       fprintf(stdout, "␣");
133     if (hr) fprintf(stdout, "␣\033[36m[\033[1;31m%02d:", hr);
134     else    fprintf(stdout, "␣\033[36m[\033[1;31m");
135     fprintf(stdout, "%02d:%02d\033[0;36m]\033[0m\r", min, sec);
136     fflush (stdout);
137     }
138 #endif//CLI_CONTROL
139 }
140
141 void play_fileinfo(int bitrate, int freq, int duration
142 #ifdef  CLI_CONTROL
143                    , int chans, char* verstr
144 #endif//CLI_CONTROL
145                    )
146 {
147     char* unk="Unknown";
148 #ifdef  GUI_CONTROL
```

```
149     char* str = playlist->pcur->id3tag.title  ?
150                 playlist->pcur->id3tag.title  : unk;
151     strncpy(shm->title +1, str, MAX_TAGINFO_SIZE-1);
152           str = playlist->pcur->id3tag.artist ?
153                 playlist->pcur->id3tag.artist : unk;
154     strncpy(shm->artist+1, str, MAX_TAGINFO_SIZE-1);
155     shm->title[0] = shm->artist[0] = 'r';
156     shm->duration = duration;
157     shm->bitrate  = bitrate;
158     shm->freq     = freq;
159 #endif//GUI_CONTROL
160 #ifdef  CLI_CONTROL
161     {   int hr=0, min=0;
162     if (duration > 3599) { hr = duration / 3600; duration %= 3600; }
163     if (duration > 59)   { min= duration / 60;   duration %= 60;   }
164     fprintf(stdout, "\033[1;32mTitle:␣␣%s\nArtist:␣%s\n"
165                     "%s,␣%dKHz,␣%dKbps(%s),␣%s(%d)\n"
166                     "Playback␣in␣progress␣......"
167                     "␣[%d:%02d:%02d]\033[0m\n",
168                     playlist->pcur->id3tag.title  ?
169                     playlist->pcur->id3tag.title  : unk,
170                     playlist->pcur->id3tag.artist ?
171                     playlist->pcur->id3tag.artist : unk,
172                     verstr, freq, bitrate<0 ? -bitrate : bitrate,
173                     bitrate<0 ? "VBR"  : "CBR",
174                     (chans==1 ? "Mono" : "Stereo"), chans,  // debug
175                     hr, min, duration);
176     }
177 #endif//CLI_CONTROL
178 }
179
180 inline void preload(FileInfo* fi)
181 {
182     if (fi == playlist->lcur)                return;
183
184     if (fi->loff < fi->size)                 playlist->lcur = fi;
185     else if (fi->next->loff < BUF_THRESHOLD) playlist->lcur = fi->next;
186     else if (fi->next->offset > (BUF_TOTAL_SIZE>>3)) {
187         fi->next->loff = fi->next->offset = 0;
188         playlist->lcur = fi->next;
189     }   else                                 return;
190 dtrace;
191     if (kill(pid_hdl, SIG_LOAD))
192         dprint("Failed␣to␣send␣signal␣to␣hdload.\n");
193 }
194
195 CmdStatus proc_cmd()
196 {
197     extern unsigned mp3_frame_len;
```

```
198     FileInfo* fi = playlist->pcur;
199     int  fbwd=0, undelay=0, frame_len=0, pause=0, cmd=CMD_INVALID;
200 #ifndef GUI_CONTROL
201     fd_set fdset;               struct timeval tv={0, 0};
202 #endif//GUI_CONTROL
203     do {
204 LOOP:
205 #ifndef GUI_CONTROL
206         FD_ZERO(&fdset);        FD_SET(STDIN_FILENO, &fdset);
207         if (select(1, &fdset, NULL, NULL, &tv)>0 &&
208                                 FD_ISSET(STDIN_FILENO, &fdset))
209             cmd = getchar();
210         else if (pause) {       usleep(SUSPEND_USEC);   continue; }
211         else if (fbwd) {        cmd = fbwd;             break;    }
212         else                    return CMD_STAT_READY;
213 #else   /* get command from GUI controlling */
214         if (shm->inform != COMM_COMMAND) {
215             if (pause) {        usleep(SUSPEND_USEC);   continue; }
216             else if (fbwd) {    cmd = fbwd;             break;    }
217             else                return CMD_STAT_READY;
218         }
219         shm->inform = COMM_INVALID;
220         while ((cmd = shm->command) == CMD_INVALID) usleep(DELAY_USEC);
221         shm->command= CMD_INVALID;
222 #endif//GUI_CONTROL
223         dprint("get␣a␣command:␣‘%c’(%d).\n", cmd, cmd);
224         if (cmd == CMD_PAUSE) {
225 #if 1   /* comment by mhfan */
226             int speed = (pause = !pause) ? 0 : 44100;
227             ioctl(dspfd, SNDCTL_DSP_SPEED, &speed);
228 #else   // refer to MPlayer libao2/ao_oss.c
229             if ((pause = !pause)) {
230                 ioctl(dspfd, SNDCTL_DSP_RESET, NULL); close(dspfd);
231             } else if ((dspfd=open(DSP_PATH, O_WRONLY)) < 0)
232                 dprint("Can’t␣open␣audio␣device:␣%s", DSP_PATH);
233 #endif  /* comment by mhfan */
234         }
235     } while (pause);
236
237     switch (cmd) {
238 #ifdef  GUI_CONTROL
239     case CMD_OVERSONG:
240     case CMD_SELECT:
241         reset_playlist(shm->first, shm->last, shm->rule);
242         playlist->pcur = playlist->fiarr + shm->selidx;
243         playlist->pcur = playlist->pcur->prev;          break;
244 #endif//GUI_CONTROL
245     case CMD_NEXT:                                      break;
246     case CMD_PREV:      playlist->sum += 2;
```

```
247         playlist->pcur = fi->prev->prev;                break;
248     case CMD_REPLAY:     playlist->sum++;
249         playlist->pcur = fi->prev;                      break;
250     case CMD_FORWARD:
251         if (! fbwd) {   fbwd = cmd;    clear_dspbuf(); undelay = 1024;
252             switch (fi->fmt) {
253             case AUDIO_MP3:
254                 play_frames -= fi->bufs.cur->frames;
255                 if (!(shm->bitrate < 0 &&
256                         (frame_len = playlist->poff / play_frames)))
257                     frame_len = mp3_frame_len;          break;
258             case AUDIO_WMA:
259                 play_frames -= fi->bufs.cur->frames;
260                 frame_len = fi->size * WMA_FRAMES_PER_THOUSAND /
261                         wma_fh->duration - 20;          break;
262                 //frame_len = wma_fh->packet_size / 5;  break;
263             case AUDIO_WAV:
264             case AUDIO_OGG:             case AUDIO_UNK:
265             }
266         }
267         if (fi->loff - playlist->poff < BUF_THRESHOLD) preload(fi);
268         fi->bufs.cur = fi->bufs.cur->next;
269         playlist->poff += BUF_MAX_SIZE;
270         if (frame_len)
271             play_frames += (fi->bufs.cur->frames = BUF_MAX_SIZE / frame_len);
272             //play_frames += (fi->bufs.cur->frames =
273             //      play_frames * BUF_MAX_SIZE / playlist->poff;
274         else {  off_t offset = playlist->poff;
275             wav_AdpcmDecode(fi->bufs.cur->base, BUF_MAX_SIZE, NULL, &offset);
276         }
277         if (!(playlist->poff+BUF_MAX_SIZE < fi->size))  return CMD_STAT_RETURN;
278         play_progress();                usleep(FBWD_DELAY - undelay);
279         if (undelay < (FBWD_DELAY>>1))  undelay <<= 1;  goto LOOP;
280     case CMD_BACKWARD:
281         if (fi->bufs.cur == fi->bufs.begin) {           playlist->sum += 2;
282             playlist->pcur = fi->prev->prev;            return CMD_STAT_RETURN;
283         }
284         if (! fbwd) {   fbwd = cmd;     clear_dspbuf(); undelay = 1024; }
285         playlist->poff -= BUF_MAX_SIZE;
286         fi->bufs.cur = fi->bufs.cur->prev;
287         play_frames -= fi->bufs.cur->frames;
288         if (fi->offset > 0) {
289             preload(fi);                usleep(SUSPEND_USEC);
290         }   play_progress();            usleep(FBWD_DELAY - undelay);
291         if (undelay < (FBWD_DELAY>>1))  undelay <<= 1;  goto LOOP;
292     case CMD_ENDFB:      fbwd = 0;                      return CMD_STAT_CONT;
293     case CMD_MEM:                                       return CMD_STAT_MEM;
294     case CMD_NMEM:  playlist->pa = playlist->pb = 0;    return CMD_STAT_NMEM;
295 #ifdef  GUI_CONTROL
```

```
296     case CMD_CHANGERULE: reset_playlist(shm->first, shm->last, shm->rule);
297                                                     return CMD_STAT_READY;
298 #endif//GUI_CONTROL
299     case CMD_QUIT:      playlist->rule=PR_SEQUENCE;
300                         playlist->sum = 1;          break;
301     default :                                       return CMD_STAT_READY;
302     }                   clear_dspbuf();             return CMD_STAT_RETURN;
303 }
304
305 void reset_playlist(int first, int last, PlayRule rule)
306 {
307     int* flag=malloc((last = last-first+1) * sizeof (int));
308     FileInfo *fi, *fi0 = playlist->fiarr + first;
309     dprint("first=%d,␣last=%d,␣rule=%d\n", first, last+first-1, rule);
310
311     srandom(time(NULL) | getpid());     // initilize a random generator.
312     if (rule==PR_SHUFFLE || rule==PR_RANLOOP) {
313         int i, *ran=malloc(last * sizeof (int));
314         for (first=0; first < last; ++first) ran[first] = 0;
315         for (first=0; first < last; ++first) {
316             while (ran[(i=random()%last)]) ;
317             flag[first] = i;     ran[i] = 1;
318         }                        free(ran);
319     } else for (first=0; first < last; ++first) flag[first] = first;
320
321     fi = playlist->lcur = fi0 + flag[0];
322     for (first=1; first < last; ++first) {
323         fi->next = fi0 + flag[first];
324         fi->next->prev = fi;
325         fi = fi->next;
326     }
327     playlist->sum  = last;
328     playlist->rule = rule;
329     fi->next = playlist->lcur;
330     playlist->lcur->prev = fi;
331     playlist->lcur = NULL;
332     free(flag);
333 dtrace;
334 }
335
336 void init_playlist(int fsum, char* path[])
337 {
338     int i;          FileInfo* fi;
339     if (! (fi = playlist->fiarr = malloc(fsum * sizeof (FileInfo))))
340         EXIT(ERR_MALLOC);
341     dprint("fsum=%d,␣path=%p\n", fsum, path);
342
343     for (i=0; i < fsum; ++i) { char* fns=strrchr(path[i], '.');
344             if (!strcasecmp(fns, ".mp3")) fi->fmt = AUDIO_MP3;
```

```
345         else if (!strcasecmp(fns, ".wma")) fi->fmt = AUDIO_WMA;
346         else if (!strcasecmp(fns, ".wav")) fi->fmt = AUDIO_WAV;
347         else if (!strcasecmp(fns, ".ogg")) fi->fmt = AUDIO_OGG;
348         else                               fi->fmt = AUDIO_UNK;
349         dprint("### playlist->path: %s(idx=%d)\n", path[i], i);
350         fi->offset = fi->loff = 0;
351         fi->size = INVALID_SIZE;
352         fi->path = path[i];
353         fi = fi + 1;
354     }
355 dtrace;
356 }
357
358 int main(int argc, char* argv[])
359 {
360     PRINT_VERSION;
361 #ifndef GUI_CONTROL
362     if (argc < 2) {
363         fprintf(stdout, "Usage: %s foo.mp3 bar.wma ...\n", argv[0]);
364         return 0;
365     }
366     init_playlist(argc-1, &argv[1]);
367     reset_playlist(0, argc-2, PR_RANLOOP);
368     playlist->pcur = playlist->fiarr;
369     playlist->lcur = NULL;
370 #else   /* get all path from GUI shared memeory */
371     shm->sign = 1;              //usleep(SUSPEND_USEC);
372     init_playlist(shm->fsum, shm->path);
373     reset_playlist(shm->first, shm->last, shm->rule);
374     playlist->pcur = playlist->fiarr + shm->selidx;
375 #endif//GUI_CONTROL
376
377     if ((pid_hdl=vfork()) < 0)  EXIT(ERR_FORK);
378     if ( pid_hdl==0 && execl(WORK_PATH HDL_NAME, HDL_NAME, NULL)<0)
379                                 EXIT(ERR_CREATE_PROC);
380
381     {   struct sigaction sar;
382     sar.sa_handler = sighdl_ready;
383     sar.sa_flags = SA_RESTART;
384     sigemptyset(&sar.sa_mask);
385     if (sigaction(SIG_READY, &sar, NULL) < 0) EXIT(ERR_SIG_INIT);
386
387     sar.sa_handler = sighdl_term;
388     if (sigaction(SIGTERM, &sar, NULL) < 0) EXIT(ERR_SIG_INIT);
389     }                           pause();
390
391     setpriority(PRIO_PROCESS, 0, -20);  // Make self the top priority process!
392
393     if ((dspfd = open(DSP_PATH,
```

```
394 #ifdef  DUMP_OUTPUTS
395                     O_WRONLY | O_CREAT
396 #else
397                     O_WRONLY//, 0660
398 #endif
399                     )) < 0)     EXIT(ERR_DSP_OPEN);
400     init_dsp(44100, 1, 16);
401
402     for (;;) {  FileInfo* fi = playlist->pcur; int  cnt;
403 #ifdef  GUI_CONTROL
404         for (cnt=0; shm->inform  == COMM_INVALID;) {
405 #ifdef  DEBUG
406             if (++cnt > MAX_WAIT_LOOP)
407                 dprint("Wait too long for GUI: %d.\n", __LINE__);
408 #endif//DEBUG
409             usleep(DELAY_USEC);
410         }       shm->inform   = COMM_INVALID;
411         for (cnt=0; shm->command == CMD_INVALID;) {
412 #ifdef  DEBUG
413             if (++cnt > MAX_WAIT_LOOP)
414                 dprint("Wait too long for GUI: %d.\n", __LINE__);
415 #endif//DEBUG
416             usleep(DELAY_USEC);
417         }       shm->inform   = COMM_INVALID;
418         if    (shm->command == CMD_QUIT)        break;
419               shm->command  = CMD_INVALID;
420 #endif//GUI_CONTROL
421 #ifdef  CLI_CONTROL
422         fprintf(stdout, "\033[1mPlaying audio file: %s(idx=%ld). begin=%p\n",
423                 fi->path, fi - playlist->fiarr, fi->bufs.begin->base);
424 #endif//CLI_CONTROL
425         if (fi->offset > (BUF_TOTAL_SIZE>>3) ||
426                 (fi->loff - fi->offset) < (BUF_TOTAL_SIZE>>3))
427             fi->loff = fi->offset = 0;
428         for (cnt=0; fi->loff < (BUF_THRESHOLD<<1) || fi->offset > 0;) {
429 #if 0   /* comment by mhfan */
430             if (cnt) {  cnt = 0;                preload(fi); }
431             usleep(SUSPEND_USEC);
432 #else
433             preload(fi);                        usleep(SUSPEND_USEC);
434 #ifdef  DEBUG
435             if (++cnt > (MAX_WAIT_LOOP>>2))
436                 dprint("Wait too long for hdload: %d.\n", __LINE__);
437 #endif//DEBUG
438 #endif  /* comment by mhfan */
439             if (fi->size == 0) { fi->size = INVALID_SIZE;       goto NEXT; }
440         }   playlist->pa = playlist->pb = playlist->poff = 0;
441         fi->bufs.cur = fi->bufs.begin;
442         switch (fi->fmt) {
```

```
443         case AUDIO_MP3:           init_mp3d();   mp3dec();        break;
444         case AUDIO_WMA:     if (!init_wmad())    wmadec();       break;
445         case AUDIO_WAV:     if (!init_wavd())    wavdec();       break;
446         case AUDIO_OGG:          //init_oggd();  oggdec();       break;
447         case AUDIO_UNK:
448             dprint("Unknown audio format: %s\n", fi->path);     break;
449         }   // waiting for over playing
450 NEXT:
451 dtrace;
452         playlist->pcur= playlist->pcur->next;
453 #ifdef  GUI_CONTROL
454         shm->playidx  = playlist->pcur - playlist->fiarr;
455 #endif//GUI_CONTROL
456         if (!(--playlist->sum)) {
457 dtrace;
458             if (playlist->rule < PR_SEQLOOP) {
459 #ifdef  GUI_CONTROL
460                 shm->feedback = COMM_EXIT;
461 #endif//GUI_CONTROL
462                 break;
463             }
464             else if (shm->rule != playlist->rule)
465 #ifdef  GUI_CONTROL
466                 reset_playlist(shm->first, shm->last, shm->rule);
467 #else
468                 reset_playlist(0, argc-2, PR_RANLOOP);
469 #endif//GUI_CONTROL
470         }
471 #ifdef  GUI_CONTROL
472         shm->feedback = COMM_OVERSONG;
473 #endif//GUI_CONTROL
474     }   sighdl_term(SIGTERM);               return 0;
475 }
476
477 /******************** End Of File: ucap.c ********************/
```