How to control madplay through button press

Madplay is a command-line MPEG audio decoder and player based on the MAD library(libmad). We tried to use madplay to control the music play operation at first, but madplay could only be controlled from keyboard. When we tried to redirect the input from stdin to a named pipe, it still did not work. Our boards do not come with a keyboard, so basically we have to find another way to replace keyboard.

To be able to control madplay operation through button press, one has two main steps to complete:

- Step one involves modifying madplay source code such that madplay gets its control command from a named pipe.
- Step two involves writing to the named pipe on button press event.

Kernel Configuration

Refer to Brian's guide for details.

Madplay source code modification

1. Download latest versions of madplay, libmad, and libid3tag from http://sourceforge.net/projects/mad/files.

2. Extract each .tar.gz file into ~/cmpt433/private/mad-src and cd into madplay folder.

3. Open up file player.c, locate function definition static enum mad_flow tty_filter(void *data, struct mad_frame *frame) somewhere around lines 2236 – 2337, observe that there is command = readkey(0) on line 2243 and command = readkey(1) on line 2263. These two readkey commands basically read control commands from keyboard. We are going to comment out these two commands and replace with command = readkey1(0) and command = readkey1(1).

4. Write a new function readkey1 as follows which would read control command from a named pipe:

```
 # define FIFO_NAME "/tmp/madplayFIFO"  // put this line near the beginning of the file.
static int readkey1(int blocking)
 {
   unsigned char key;
   int pipe, res;
   ssize_t count;
   // check whether named pipe exists, if not, create it.
   if (access(FIFO_NAME, F_OK) == -1) {
    res = mkfifo(FIFO_NAME, 0777);
    if (res != 0)
         exit(EXIT_FAILURE);
  }
   if (!blocking) {
  // open a named pipe in nonblocking mode
    pipe = open(FIFO_NAME, O_RDONLY | O_NONBLOCK);
```

```
    count = read(pipe, &key, 1);
    close(pipe);
  }else{
  //open the named pipe in blocking mode when madplay in pause mode.
    pipe = open(FIFO_NAME, O_RDONLY);
    count = read(pipe, &key, 1);
    close(pipe);
  }
  return (count == 1) ? key : 0;
}
```

5. Now we can compile libmad, id3tag and madplay application. (Refer to Brian's guide for details).

Button Press

1. Create a folder buttonControlMadplay. Since Button press needs to include module buttondrv, I added my own version: pushbuttondrv.h and pushbuttondrv.c into this folder.

2. Create a file buttonControlMadplay.c with following code:

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "pushbuttondrv.h"
#define FIFO_NAME "/tmp/madplayFIFO"
#define FALSE 0
#define TRUE 1
int pipe1;
void pause1()
{
  pipe1 = open(FIFO_NAME, O_WRONLY);
  write(pipe1, "p", 1);
  printf("pause\n");
  close(pipe1);
}
void resume()
{
  pipe1 = open(FIFO_NAME, O_WRONLY);
  write(pipe1, "o", 1);
  printf("resume\n");
  close(pipe1);
```

```
}
void forward()
{
  pipe1 = open(FIFO_NAME, O_WRONLY);
  write(pipe1, "f", 1);
  printf("forward\n");
  close(pipe1);
}
void backward()
{
  pipe1 = open(FIFO_NAME, O_WRONLY);
  write(pipe1, "b", 1);
  printf("backward\n");
  close(pipe1);
}
void increasevolume()
{
 pipe1 = open(FIFO_NAME, O_WRONLY);
 write(pipe1, "+", 1);
 printf("increase volume\n");
 close(pipe1);
}
void reducevolume()
{
 pipe1 = open(FIFO_NAME, O_WRONLY);
 write(pipe1, "-", 1);
 printf("reduce volume\n");
 close(pipe1);
}
void quit()
{
 pipe1 = open(FIFO_NAME, O_WRONLY);
 write(pipe1, "q", 1);
 printf("quit\n");
 close(pipe1);
}
int main()
{
 int buttonPressed;
 int paused = FALSE;
 PUSHBUTTONDrv_init();
 while(TRUE){
  buttonPressed = PUSHBUTTONDrv_detect();
  printf("button pressed is %d\n", buttonPressed);
  if (!paused){
    if (buttonPressed == 9) quit();
    if (buttonPressed & RIGHT_BUTTON_REP) forward();
    if (buttonPressed & LEFT_BUTTON_REP) backward();
```

```
    if (buttonPressed == 6){
          pause1();
          paused = TRUE;
    }else if (buttonPressed & DOWN_BUTTON_REP)
          reducevolume();
    else if (buttonPressed & UP_BUTTON_REP)
          increasevolume();
   }else{
    if (buttonPressed == 6){
          resume();
          paused = FALSE;
    }
   }
  }
  PUSHBUTTONDrv_cleanup();
}
```

3. Add a Makefile to automate the compile process and compile the buttonControlMadplay.

```
CC=arm-linux-gcc
EXECUTABLE=buttonControlmadplay
CFLAGS=-Wall -g
COMPILE=$(CC) $(CFLAGS) -c

SRCS:=$(wildcard *.c)
OBJS:=$(patsubst %.c, %.o, $(SRCS))

$(EXECUTABLE):$(OBJS)
      $(CC) -o $(EXECUTABLE) $(OBJS)
      -cp $(EXECUTABLE) ${PUBLIC}/bin

%.o: %.c
      $(COMPILE) -o $@ $<
```

Testing

1. To run madplay on your target via NFS . Refer to Brian's guide for details.

2. To run buttonControlMadplay
   insmod EmbedSky_irq.ko
   ./buttonControlMadplay

3. Press right button, madplay should move forward.
   - Press left button, madplay should move back.
   - Press up button, madplay increases volume.
   - Press down button, madplay reduces volume.
   - Press left and right button simultaneously, madplay should pause.
   - Press left and right button simultaneously once more, madplay should resume.
   - Press up and down buttons simultaneously, madplay should quit.

Troubleshooting

- "Madlib.so not found", run printenv to verify that you have correct LD_LIBRARY_PATH. If not, run export LD_LIBRARY_PATH=/mnt/remote/mad/lib:$LD_LIBRARY_PATH on the target.
- "Can not open Irq button", run lsmod to make sure that you have insmodded EmbedSky_irq.ko module.