Alan Achtenberg

# Lab 9 AC97 Codec Device Driver

ECEN 449 Sec:505

Due: 12/5/2014

Introduction:

In Lab 9 we implemented a device driver for the AC97 audio hardware. The set up for the hardware was done previously in Lab 8, which allowed us to focus on just creating a kernel module for our device as well as a user space devtest program.

Procedure:

As I said before the hardware was previously set up, so the first step of Lab 9 was to create a new kernel module. Starting from the device driver we wrote in lab 8 as a template. We disabled the read function. We used ioremap to get the virtual address of the AC97 bus. We added the necessary functions in the open method to initialize the AC97 hardware and register an interrupt handler for the device. We implemented the device release method, to release all the resources and unmap the device file. Finally we wrote our interrupt handler to simply check if the in fifo of our device is full. If it is not full we added samples to the buffer from audio_samples.h in a circular fashion. In other words when we got to the end of the samples we simply repeated the sequence.

The second part of Lab 9 was to create an IOCTL function for our device driver. This function allows the user to send short commands to the IO device allowing it to change configuration as well as request information about the device itself. First we added the function to our file operation structure, FOPS. Then we implemented a simple function that could set the auxiliary volume(headphone volume), master volume, or the playback speed. These values are set by passing in an operation number (conveniently defined in sound.h) and a value by reference. Of course we must use the getuser function to check this value.

Finally after completing our device driver we wrote a simple detest that would open the file and then periodically change the settings of the device.

Results: C Source Files

Conclusion:

Lab 9 was in theory harder to implement then lab 8, but since the hardware creation process and the linux kernel process was already complete, it actually took much less time to implement. The device driver greatly enhances the simple capabilities of the device.

Questions:

(a) The ioctl() interface is slowly becoming depreciated in the Linux kernel. Why are kernel developers moving away from it? Outline a different method to achieve the same end.

The ioctrl() interface can frequently need to be called, if it is called frequently enough it can bottleneck your entire system, because the ioctrl function locks the kernel. A better way to implement this would be to use memory mapped files.

(b) What would be the effect of using a small, 16 word buffer as in the original AC'97 device?

The interrupt handler would be called pretty much every cycle. Greatly increasing the overhead and number of context switches. The device driver would not be able to keep up with the demand of the device.

(c) Currently the interrupt trigger point is set to half empty on the playback buffer. What might be the consequences of lowering the trigger point? What would be the effects of raising the trigger point?

If you lower the point (less than half full) you run the risk of your device having to buffer before it can play audio. In other words you might hear skipping(periods of time with no audio). If you raise the point you run the risk of allowing the interrupt handler to hog the cpu and starve out other processes.

(d) In your current audio device driver, a significant amount of data transfer is being performed within the interrupt handler. What problems could this cause? Skim through chapter 10 of Linux Device Drivers, Third Edition and provide a solution to the existing problem.

The interrupt handler is likely to run for a significant amount of time, this might allow another interrupt signal to be received before the handler has completed. This could cause errors within the buffering action of the device.