Lab 6: Musical Keyboard

Objective

In this experiment, students will be exposed to:

- Different approaches for creating periodic tasks in Kernel space.
- Creating ISR's to handle GPIO interrupts on the Raspberry Pi 3 board.
- Learn about communication between Kernel space and User space
- Use the Master/Slave setup from Lab 5 to control the musical keyboard.

Prelab

For this prelab, you should investigate and include a brief description for the following functions: request_irq, free_irq, hrtimer_init, kthread_create, kthread_stop.
You should look at *section 6: General Purpose I/O* in the manual: *BCM2835-ARM-Peripherals* (on Canvas under *Modules > Other Material*). Pay attention to the registers associated to configuring and detecting level and edge events. You will need those for this lab.

Lab Procedure

Part 1: Speaker function

For this part of the lab, you are to create a module that contains a "speaker function". The purpose of the speaker function is to create a square wave that will activate the speaker on the auxiliary board. The speaker is connected to GPIO 6 on the RPi.
You will try two different approaches, illustrated by the following examples:

a) **kthread_test.c:** This module implements a "Kernel thread". Examine the code, then compile and install it. Use the `dmesg` command to see the messages printed.
   Use this example as a basis for a kernel thread based speaker function. Experiment with different delays until you hear a nice sound. You may keep the printk statements for debugging purposes, but you should remove them eventually. You should not keep printk's inside loops that go relatively fast.
b) **hrtimer_test.c:** This module implements a "high resolution timer". As before, examine the code, compile and install it. Check the messages printed using `dmesg`.
   Use this example to create your own timer for the speaker.

Post lab Questions: Based on your observations, which approach did you like better? Did either of them produce a better sound? Which one do you think is more reliable/accurate?

Part 2: The Keyboard

For this part of the lab you will add an Interrupt Service Routine (ISR) to your modules from Part 1. The purpose of the ISR is to handle one of five events, which correspond to the five push buttons on the auxiliary board. The corresponding GPIO ports and associated registers should be configured so that pushing any of the buttons causes an interrupt. The handler will change the frequency of the square wave being played by the speaker function. You will associate a different frequency to each of the buttons (lower frequency associated to button 1, highest frequency associated to button 5). Make sure that the sound produced for each case is distinct. You should configure these interrupts to be rising edge sensitive, so program the corresponding GPIO registers accordingly.

Check the file **ece4220lab6_isr.c**. It is a template that illustrates the basic steps needed to configure an interrupt and attach a handler.

Part 3: Master/Slave Control

For this part, you are to re-use your code from lab 5 to determine a master slave relationship with the other students' boards in the lab. This time, however, the master board sends the current note that it is playing to all the slave boards, so that all of them play the same note as the master's board. This requires a few steps:

1. You must program your master/slave server program to accept messages that begin with @. These messages represent one of the five notes to be played: @A, @B, @C, @D, @E.

2. If your board is a slave and it receives one of those messages, it must change the frequency of the note being played. To do this, your user space program needs to communicate with the Kernel module that you created before. Your module should still change the notes when the buttons are pressed. In other words, your module should handle the GPIO interrupt AND be able to get data from user space.

3. If your board is a master and it receives a note message (e.g. from the client program), it must also change the frequency of the note being played. Furthermore, it should "forward" that message to all the slave boards in the network.

Check the files **Lab6_cdev_kmod.c** and **Lab6_cdev_user.c**. The first one is a Kernel module that shows how to create a "Character Device", which is one way of achieving communication between kernel and user space. The module code also shows the functions that will be invoked when reading the Character Device and writing to it from user space. The second file is a template for a user space program that opens and uses the Character Device. Use these files as a reference for your programs.

Hint:

A test program named **Lab6_client** is provided (on Canvas). This program is similar to the one used for lab 5. In addition to the functionality of Lab5_client, this new client can also be used to send messages to specific IP's, not only broadcast messages. To do that, type: & #
where # is the last number of the IP address that you want to send to. For example: & 12
After you enter that, you will be asked to enter the message that you want to send. In the previous example, whatever you type after you enter & 12, will be sent to IP 128.206.19.12 (including the 'return' character).
As before, anything else you type (except if it starts with & or !) will be broadcast. And if you type '!' (without quotes) you exit the client program.
You should use this new client program to test your Lab6 server. It will be used on the post-lab due date to verify your programs.

Extra part (optional, for extra credit):

Modify your programs so that, if your board is the master, and a button is pressed, it sends the corresponding note to all the slave boards in the network.

Note 1:

There are a few other ways to achieve communication between User Space and Kernel Space. Some of those may be more efficient than the Character Device, but may be more complex to set up. You may investigate other options and implement one (or more) of them for extra credit in the project.

Note 2:

WiringPi provides a way to configure GPIO interrupts in user space on the RPi. You may explore that approach for your project.