

CSIS0230A Principle of Operating Systems (Class A)

Assignment 2

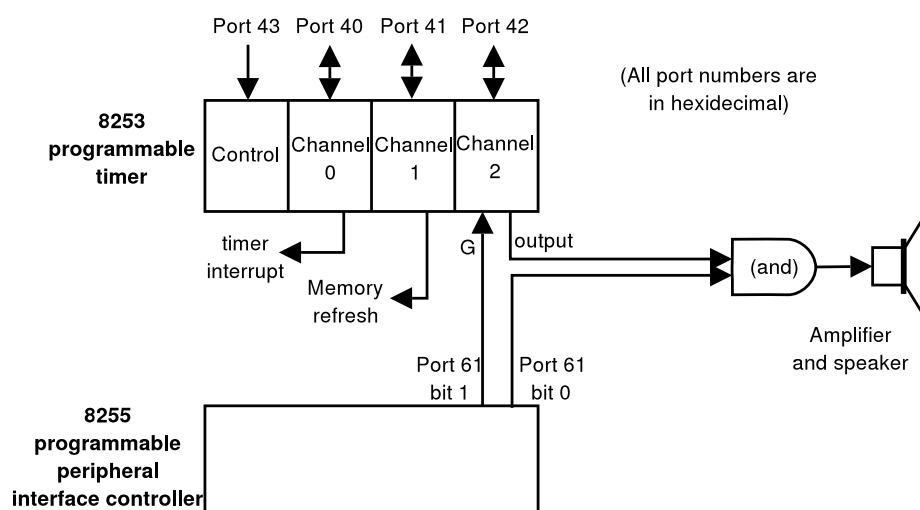
Musical note generation device driver

Tutor: cktsang@csis.hku.hk, Deadline Noon Oct 31, 2003.

All IBM-PC compatible computers are equipped with a speaker that is usually used to produce sound to alert the user when something wrong happens. However, the speaker is capable of producing sound of any frequency. In this assignment you will ask it to produce musical notes, by creating a device which converts characters into frequencies and put it into the speaker.

1. The circuitry controlling the PC-speaker

In the era when 8086 and 8088 were the main players in the PC world, the PC-speaker is controlled by two ICs called 8253 (programmable timer) and 8255 (programmable peripheral interface controller). Nowadays, they are replaced by an “I/O chipset”, which provides the functionalities of the many chips. But the programming interface is unchanged, so we will still use the same name to describe them. They are connected as follows:



There are two ways to control the speaker: one can either disable the timer (so that the output is stuck at 1) and write 0 and 1 into bit 0 of port 0x61 at predefined time; or program the timer to switch between 0 and 1 periodically and leave bit 0 of port 0x61 to be 1. In either way, a waveform is produced, driving the speaker to produce sound. The former requires constant attention by the CPU, while the latter does not use CPU but can only produce single frequency square wave. For our assignment, we use the latter method (i.e., use the 8253 timer chip).

The 8253 chip is the main timer chip of the computer. It contains 3 timers, or “channels”, numbered 0, 1 and 2. Each can be programmed independently. Channel 0, 1 and 2 are used for producing timer interrupts, refreshing the memory¹, and driving the speaker. Our concern is of course channel 2.

The 8253 chip is connected to an oscillator of 1.193MHz. The timer works by counting cycles of this oscillator, and the G input in the figure starts and stops the counting. The number of cycles to count is controlled by an internal 16-bit counter of the 8253 chip, latched into the timer through

¹Memory are called “Dynamic RAM” because they work by storing electricity in capacitors, which leaks and loses data if not “refreshed” (i.e., have the same value reloaded) for too long. Nowadays memory refresh circuitry is within RAM modules themselves.

port 0x42. However, one must follow a specific protocol to write to port 0x42, otherwise the timer will ignore whatever you write. In particular, before writing to port 0x42, you must write to the control port (0x43), asking it to setup channel 2 to “mode 3” with a “binary” 16-bit value. After that you have to write the counter to port 0x42 as 2 bytes, first the low-order byte of the counter value, and next the high-order byte. So the procedure to turn on the speaker is:

- Disable interrupts.
- Write to port 0x43 the control word 0xB6 (in binary, 10110110, the first 2 bits (10) select channel 2, next 2 bits (11) requests setting both bytes of the counter, next 3 bits (011) select mode 3, and the final bit (0) select binary mode).
- Write to port 0x42 the low-order byte of the counter.
- Write to port 0x42 the high-order byte of the counter.
- Read the value of port 0x61, set both bit-0 and bit-1 (using bitwise or), and write it back to port 0x61. All other bits must not be changed, or the computer might stop functioning.
- Enable interrupts.

Now the speaker should start producing sound. To stop it, do the following:

- Disable interrupts.
- Read the previous value of port 0x61. Clear both bit-0 and bit-1 (using bitwise and), and write it back to port 0x61. Again, all other bits must remain unchanged.
- Enable interrupts.

What counter value to set, then? In mode 3, the timer keeps the output high half of the time and low half of the time. So to produce a note (square wave) of h Hz, calculate $1.193M/h$ and set the quotient as the counter value. E.g., if you want to produce sound at 261 Hz (middle C), the counter should be $1.193M/261 = 4570 = 0x11DA$. So the first value to put to port 0x42 is 0xDA (i.e., 218), and the second is 0x11 (i.e., 17).

2. Device description

Write a Linux device driver to implement a device. The device can only be written into, not read from. When a character is written to the device, the behaviour depends on the character:

- 1–7: produce the note C, D, E, F, G, A and B of the current octave, unless modified by b or #. The note should be of one note-length, unless modified by < or >.
- 0: produce no sound for one note-length, unless modified by < or > as specified below.
- # and b: change the next number (0 to 7) so that it produce sound one semitone higher and lower, respectively. If more than one such character precedes a number, only the last one has a effect. The character has no effect if the next number is 0.
- < and >: half or double the length of the next number. The effect is culmulative, i.e., 2 <'s decrease the length to 1/4 of the original. But the effect should last only for 1 note.
- ^ and v: change the current octave to the lower and higher octave. There should be 5 octaves; when the device is opened the current octave starts at the 3rd (i.e., the middle one, which is said to be the base octave). If it results in an octave out of possible range, the ^ or v character should be ignored.
- All other characters should be silently ignored.

The device should also support 2 *ioctl()*'s: one to read the current note-length, and one to set it. You must design the interface to this *ioctl()*'s yourselves, and write a program `notegen-length` which utilize the *ioctl()*'s to get and set the speed of the note generation device.

Here are some more technical requirements of the device:

- The device driver should be implemented in a kernel module. The kernel module should not allocate any I/O port, since the ports needed are allocated by the system already.
- By default, the major number of the device should be dynamically allocated. However, there should be a module parameter `devmajor` allowing the user to specify the major number for it. Failure to allocate the major number should result in the error code *EBUSY*.
- You should only allow the device to be opened for writing, not for reading. The open mode should be checked when the device is opened. Opening in other modes should result in the error *EINVAL*.
- The device can be opened by only one process at a time. If an attempt is made to open the device when the device is already in use, it should result in the error code *EBUSY*.
- At any time, only one operation (*ioctl* or *write*) of a device can be active. E.g., if a thread of a process is writing to a device, the second thread cannot write to the device again. Any such attempt should fail with *EBUSY*.
- Since many user programs fail to call the *write()* system call again in case the write is completed only partially, the write method of the device should output the complete buffer to the device before returning (unless a signal interrupts it, see below).
- It should be possible to interrupt a pending write to the note-generation device by a signal. An interrupt should be checked (using *signal_pending()*, see ULK Chapter 9 p. 256) when a complete character is generated. In the middle of note-generation, the process should be in an “uninterruptable” state.
- Initially, the length of a note should be 500 milliseconds. This duration should be limited to between 20 milliseconds and 1000 milliseconds. An attempt to set the note-length outside this range should instead set it to the limit. (E.g., an attempt to set it to 10 milliseconds should actually set it to 20 milliseconds.)

3. Testing out your device

You should write programs to test your device for various conditions. However, as a preliminary test, you may use `echo` and `cat` to test your device. E.g., the following should ask the speaker to output a few notes of a folk song:

```
/bin/echo '^>112>334>565>30>543>20>432>>1' > notegen
```

where `notegen` is a device you make for note-generation. Remember to type the `/bin/`: the `bash` shell program provides an internal command called `echo` which will be invoked if you don't type the `/bin/` prefix, but the shell ignores signals so you will be unable to test whether a signal can interrupt writing to the device.

4. Handin

Send the module source file, named `notegen.c`; and the source of the user program `notegen-`

length, named `notegen-length.c`. In the comment of the latter file, explain the usage of the command, and also the `ioctl`'s that set and get the note length. You should also write user programs to test your module, but you do not need to handin those user programs.

This is a group assignment, allowing at most 2 students in a group. Put the names, university numbers and login-ids of all group members as comment of the code you hand-in.

5. References

Tutorial 6 notes and solution. (Device driver.)

Linux Device Drivers, 2nd Edition. (<http://www.xml.com/ldd/chapter/book/>):

- Chapter 2: I/O Ports and I/O Memory; read and write.
- Chapter 3: File operations.
- Chapter 5: `ioctl`, Access Control on a Device File.
- Chapter 6: Delaying Execution.
- Chapter 8: Using I/O Ports.
- Chapter 9: Overall control of interrupts.

Note: in many places of the book you will notice references to “spinlocks” and “semaphores”. They makes sure devices work properly when loaded in a computer with multiple processors. Our course didn't have covered them yet, so you may opt to ignore all these issues.

6. Frequencies of notes

The frequencies of the notes in the base octave are as follows. A semitone is the difference between two consecutive lines in the following table.

Note	Frequency
C	261
C#	276
D	293
D#	310
E	329
F	348
F#	369
G	391
G#	414
A	439
A#	465
B	493

Each increment in octave doubles the frequency, while each decrement in octave halves the frequency. This can easily be implemented by bit shifting operations.