Midi Sampling, last revised on 8 Jul 2008

Worksheet – Midi Processing > Midi Sampling

Goals:

-To create a midi player that can play either a single note or a chord based on user input.
(sample and introduction assume a Linux based operating system)

Introduction:

Sound is digitally represented in many ways. The most common way uses sampling to approximate the shape of the wave. However, this process consumes a lot of memory, and more importantly bandwidth during transfers. This level of accuracy allows for an nearly exact replica to be played on the other end. Sometimes it is only necessary to transfer one generic type of sound, such as an piano solo. Since all pianos play basically the same sound, it should be possible to pass only what an artist does to the piano (keys pressed, how fast, and when) and the fact that the instrument is a piano. A system of data organization was developed by leaders in the digital music industry to allow this. At the end of this worksheet is a table showing how this data is encoded, but only the note on command is needed for this lab. The default note on command byte is 0x90. A full command is composed of three bytes:
Command Byte: 0x90 Note Byte: 0 – 0x7F Velocity Byte: 0 – 0x7F
The note byte contains the note that is played. The benchmark note for this is C3 which is conveniently located at 0x3C. The velocity byte contains the rate at which the key was pushed and is normally used as an amplitude multiplier. For simplicities sake, we will ignore it's amplitude implications for this assignment. However the velocity contains one important use. Most synthesizers use a velocity of zero to mean a note off command. This is something you should keep in mind. Midi notes can be converted to a fundamental frequency by a table or more easily with the following equation:
freq = 8.1758*2^ (midi/12.0)

Another tool you may need for this assignment is /dev/dsp. This device controls audio input and output. BEWARE: This device can only be accessed by one process at a time. This means you should close all other audio programs before accessing it, and you must close it. If you fail to close it before your program exits, no other process will be able to use the dsp. This means you will have to restart your computer before continuing. I personally suggest the following code as a safeguard:

```
typedef void (*fptr)(int signum);  // prototype a function pointer

FILE *fd;  // declare file pointer (or integer) globally

int main(...){
 .......
 signal(SIGINT, (fptr)endLoop); // this will call the function endLoop when you
                                // send a CTRL+C interrupt from the keyboard
 .......
}

// The following code closes the file pointer and then exits the program
```

```
void endLoop(int signum){
 fclose(fd); // or close(fd) if an using open() to open
 exit(0);
}
```

By writing data to the dsp, you can make the computer play sounds. The default settings of the dsp assume a 8 bit data width or one character and a 8 kHz sampling rate. While in any real situation it is wise to manually set these parameters, your computers are probably generic enough to survive an assumption (and this introduction is getting really long).

To get a midi note into a format that the dsp can play it is necessary to create a sampling effect. This can be done by converting the frequency into a sin wave and sampling. 8000 samples are necessary for one second of sound. The sin wave needs to have a frequency of the midi note. Use the following equation to your advantage:

sin(f*t)

f is the frequency and t is the time. The time needs to move from 0 – 2π in 8000 intervals. I leave the how to of this to you.

This will give you data from -1 to 1, but your final data needs to be in unsigned character form from 0 – 255.

So in summary to play a midi note or chord:

Tasks:

- Have user decide the number of notes to play
- Have user input notes
- Convert notes to midi
- Convert each midi to a frequency
- Convert the frequency to a sampling
- Add the samples of all the notes together
- Scale the sampling to an array of unsigned characters between 0 and 255
- Write the array to the /dev/dsp

Submit your program using standard procedures.

Additional Information:

Midi Command	Base Hex Byte (128-255)	Data Byte 1(0 - 127)	Data Byte 2 (0-127)
Note Off*	0x80	Key #	Off Velocity
Note On	0x90	Key #	On Velocity
Poly Key Pressure	0xA0	Key #	Pressure value
Control Change	0xB0	Control #	Control Value
Program Change	0xC0	Program #	--Not Used--
Mono Key Pressure	0xD0	Pressure value(0-127)	--Not Used--
Pitch Bend	0xE0	Range(LSB)	Range(MSB)
System	0xF0	**	**

* Note Offs are rarely used, instead Note Ons with a velocity of zero are used.
**System commands are designed by the manufacturer, so their data bytes could be anything
However 0xFF 0x2F 0x00 is commonly used as a file end.

The least significant nibble in a command byte indicates the current channel. This value can range between 0x0 and 0xF, but represents channels 1-16.