Metronome Resource Manager

Introduction

The program operates a metronome using timers, and accepts pulses to pause the metronome for a number of seconds. A functional requirement has the metronome pause, which is done from the console using the **echo** command to send (i.e. write) **pause 4** to the metronome device:

 # echo pause 4 > /dev/local/metronome

This means that the metronome will be implemented as a QNX resource manager for the "/dev/local/metronome" device. The resource manager code (i.e., the io_write(…) function) should send a pulse to the main thread of the metronome to have the metronome thread pause for the specified number of seconds. The "pause x" should pause the metronome for x seconds, where x is any integer value from 1 through 9, inclusive. Notice the metronome resmgr is multi-threaded: main thread (resmgr) and metronome thread (interval timer).

The **metronome** program accepts three parameters from the command-line:

 # metronome beats-per-minute time-signature-top time-signature-bottom

For example: **# metronome 100 2 4**

With output:

|1&2&

|1&2&

|1&2&

|…

(the "|1" characters occur every 2*60/100 = 1.2 sec per measure)

Another example: **# metronome 200 5 4**

With output:

|1&2&3&4-5-

|1&2&3&4-5-

|1&2&3&4-5-

|…

(the "|1" characters occur every 5*60/200 = 1.5 sec per measure)

Display a usage message and terminate with failure if the exact number (not fewer; not greater; exactly) of command-line arguments is not received.

The important thing is to have the output appearing with the correct timing. **metronome** should output a pattern of single characters at the rate given by the beats-per-minute parameter, when combined with the number of intervals within each beat as shown in the table below.

Time-signature-top	Time-signature-bottom	Number of Intervals within each beat	Pattern for Intervals within Each Beat
2	4	4	|1&2&
3	4	6	|1&2&3&
4	4	8	|1&2&3&4&
5	4	10	|1&2&3&4-5-
3	8	6	|1-2-3-
6	8	6	|1&a2&a
9	8	9	|1&a2&a3&a
12	8	12	|1&a2&a3&a4&a

Here is an example calculation

For you to make sure you get the math correct.

(The table gives the details for the number of intervals based on the time signature mappings. I have highlighted the row for the example)

Given the command: **#metronome 120 2 4**

The metronome would output 120 beats per minute (➔ 60 sec / 120 beats = 0.5 sec / beat). Now using the "time-signature-top" parameter of 2, this means that there will be 0.5 sec/beat * 2 beat/measure = 1 second per measure. This means that each pattern should start on a new-line every 1 sec. Then from the table we lookup that given the top and bottom parameters, each measure happens to have 4 outputs (**intervals** from the table; not ts-bottom) with the values "|1", "&", "2", and "&". This gives (1 sec) / (4 intervals) = 0.25 sec / interval. This is the final timer setting between outputs. This means the outputs for this command will get output a 0.25 sec spacing. Your basic timer for the outputs would be set at 0.25 secs to get this spacing.

When a client reads from the metronome device (i.e. cat /dev/local/metronome), the metronome resmgr displays an information (aka info) line about the metronome to standard output (i.e. stdout). The information line is formatted as:

[metronome:<bpm> beats/min, time signature <ts-top>/<ts-bottom>, secs-per- interval: <sec/ interval >, nanoSecs: <nanSecs>]

When you enter the following command to read the status of the metronome resmgr:

cat /dev/local/metronome

You should see (assuming the metronome was started as 120 4 4):

[metronome: 120 beats/min, time signature 4/4, sec-per- interval: 0.25, nanoSecs: 250000000]

When a client writes **pause <int>** to the metronome device (/dev/local/metronome), the metronome pauses for <int> seconds, and then resumes running:

echo pause 5 > /dev/local/metronome

The domain range for <int> is: 1 – 9 (inclusive). Print an error message if <int> fails the range check, and do not terminate the metronome (i.e. the metronome continues to run on bad <int>).

For maximum marks, the metronome is to resume on the next *beat* (and not the next measure, which is |1). For example:

|1&2<pause 5>&3...

When a client writes **quit** to the metronome device (/dev/local/metronome), the metronome resmgr gracefully terminates. Remember to gracefully delete any timer(s), cancel any thread(s), close any channels, and detach from any channels.

echo quit > /dev/local/metronome

Verify the metronome resmgr is no longer running:

pidin | grep metronome

Nothing should be returned by the above command.

Acceptance Test

You (and your partner(s)) will record a movie (i.e. screencast; but not via Zoom) of the following acceptance test:

a) ./**metronome**
   Expected: usage message
b) **./metronome 120 2 4**
   Expected: 1 measure per second. I will use this unit-test to verify the correct cadence of your metronome.
c) **cat /dev/local/metronome**
   Expected: [metronome: 120 beats/min, time signature 2/4, secs-per-interval: 0.25, nanoSecs: 250000000]
d) **./metronome 100 2 4**
   Display output from: cat /dev/local/metronome
e) **cat /dev/local/metronome # displays the metronome's info**
   Expected: [metronome: 100 beats/min, time signature 2/4, secs-per- interval: 0.30, nanoSecs: 300000000]
f) **./metronome 200 5 4**
   Display output from: cat /dev/local/metronome
g) **cat /dev/local/metronome**
   Expected: [metronome: 200 beats/min, time signature 5/4, secs-per- interval: 0.15, nanoSecs: 150000000]
h) **echo pause 3 > /dev/local/metronome**
   Expected: metronome continues on next beat (not next measure)
   It's your burden to pause the metronome mid-measure (i.e. repeat until expected behaviour)
   You can be called upon to pause your metronome at any (i.e. random) point during the demo
i) **echo pause 10 > /dev/local/metronome**
   Expected: properly formatted error message, and metronome continues to run
j) **echo bogus > /dev/local/metronome**
   Expected: properly formatted error message, and metronome continues to run
k) **echo quit > /dev/local/metronome && pidin | grep metronome**
   Expected: metronome gracefully terminates