CST8244 – Real-Time Programming

Assignment #2

Introduction

You (and optionally a partner) will be building one program. The program operates a metronome using timers, and accepts pulses to pause the metronome for a number of seconds. To avoid noisy chaos in the lab, we'll use visual output for the "click" of the metronome. A functional requirement has the metronome pause, which is done from the console using a command:

 # 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 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.

The program **metronome** 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&**<nl>**|1&2&**<nl>**|1&2&**<nl>**|1&2&**<nl>**|1&2& (…)

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

Or

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

With output: # |1&2&3&4-5-**<nl>**|1&2&3&4-5-**<nl>**|1&2&3&4-5-**<nl>**|1&2&3&4-5-**<nl>**|1&2&3&4-5-**<nl>** (…)

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

Display a usage message and terminate with failure if the exact number 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

To make this part "easier", here is an example data structure that you could cut/paste/edit to your needs.

```
DataTableRow t[] = {
            {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"}};
```

As shown in the table, the metronome should output a '|' character followed by a '1' at the beginning of each measure. It should then delay for the interval length (depending on the beats-per-minute and the number of intervals per beat). At the end of each interval, it should output the next symbol in the pattern for that time signature, restarting from '|' and '1' at the beginning of the next measure.

For the purposes of this assignment, the metronome is to output one measure per line. That is, each line will begin with: |1

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 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) 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 at 0.25 sec spacing. Your timer used to drive the metronome would be set at 0.25 secs to get this spacing.

Metronome Resource Manager API

Your implementation of the metronome resource manager (resmgr) is to support the following API (in alphabetical order):

- info
- pause <int>
- quit

When a client writes **info** to the metronome device (/dev/local/metronome), the metronome resmgr writes an information line to the device. The information line is formatted as:

> metronome [<bpm> beats/min, time signature <ts-top>/<ts-bottom>

For example, use the echo command and redirection (>) to write info to the metronome device:

> echo info > /dev/local/metronome

Next, the metronome resmgr writes the information line to the device. Use the cat command to view:

> cat /dev/local/metronome

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

> metronome [120 beats/min, time signature 4/4]

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.

API Error Handling

The metronome resource manager only supports the above input commands: info, pause <int> and quit. All other inputs written to the metronome device are considered bogus (i.e. bad input). For bogus input that is written to the device, display this formatted error message to standard error (stderr):

```
Error – '%s' is not a valid command
```

where %s is the input command received from the client

For example:

```
echo this_is_NOT_supported > /dev/local/metronome
```

Displays the error message to standard error:

```
Error – 'this_is_NOT_supported' is not a valid command
```

Important: the metronome continues to run on bogus input

Marking Scheme

This assignment is worth 50 marks.

- 15 marks for metronome behaviour
- 10 marks for pause <int>
- 5 marks for info
- 5 marks for quit
- 5 marks for API error handling and usage message
- 10 marks for demonstration by screencast (i.e. movie)

Your grade will be capped at 5/15 for the behaviour and 5/10 for the demonstration if your metronome's runtime behaviour is incorrect. If your metronome can't tic-toc correctly, you're capped at 5 + 5. You're eligible for full marks on the remaining items.

Late Penalties

Good news… I'm happy to accept late assignment submissions

Bad news… a late penalty will be applied

- –10% per day to a maximum of 3 days (i.e. 72 hours = 3 days X 24 hours/day)
- –100% after 3 days

Do you regularly attend lab? If so, you may qualify for a small discount on your penalty. Let's negotiate.

Re-Submission Fee

It's past the assignment due date and time, and you just remembered… you forgot to pack something in your zip-file.

A penalty of -50% of the re-submitted item will be applied. For example, you forgot to include the screencast video. The demonstration is worth 10 points, so a re-submission penalty -5 will be applied.

Deliverables

Prepare a zip-file that contents the following items:

1. Make a screencast(s) (i.e. movie) that captures the run-time behaviour of the following scenarios:

    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. **./metronome 100 2 4**

    d. **./metronome 200 5 4**

    e. **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.

    f. **echo pause 10 > /dev/local/metronome**
    Expected: properly formatted error message, and metronome continues to run.

    g. **echo info > /dev/local/metronome && cat /dev/local/metronome**
    Expected: properly formatted info message.

    h. **echo bogus > /dev/local/metronome**
    Expected: properly formatted error message, and metronome continues to run.

    i. **echo quit > /dev/local/metronome && pidin | grep metronome**
    Expected: metronome gracefully terminates.

    Why a screencast? So I can verify the correct cadence of your metronome ☺

    You can make one screencast that captures everything, or produce many screencasts.

2. Export your Momentics IDE project as a zip-archive file.

3. A “README.txt” file reporting the status of your assignment. Follow this template:

   Title {give your work a title}

   Author
   {Please sign your work. For example: @author Gerald.Hurdle@AlgonquinCollege.com}
   Include your partner’s name (if you have one)

   {IF you worked with a partner, provide a brief description of what you worked on, and contributed to the assignment}

   Status

   {Tell me the status of your project: complete, missing requirements, not working, etc.}

   Known Issues
   {Tell me of any known issues that you’ve encountered.}

   Expected Grade
   {Tell me your expected grade.}

4. Name your zip-file according to your Algonquin College username:
   cst8244_assign2_yourUsername.zip

   For Example, cst8244_assign2_bond0007.zip

5. Upload and submit your zip-file to Brightspace before the due date.