CST8244 – Real-Time Programming

Assignment #2 – Metronome Resource Manager

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 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

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.

Important: for the purposes of the 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 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.

Metronome Resource Manager API

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

- cat /dev/local/metronome
- cat /dev/local/metronome-help
- pause [1-9]
- quit
- set <bpm> <ts-top> <ts-bottom>
- start
- stop

cat /dev/local/metronome

When a client reads from the metronome device (i.e. cat /dev/local/metronome), the metronome resmgr displays the metronome's settings to standard output (i.e. stdout) following this template:

```
[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]
```

cat /dev/local/metronome-help

When a client reads from the metronome's help device (i.e. /dev/local/metronome-help), the metronome resmgr displays information about the metronome's API to standard output (i.e. stdout). The metronome-help device is strictly read-only, so ignore any writes to the device. For example, ignore: echo Ignore writes to metronome-help > /dev/local/metronome-help

When you enter the following command to get help on the metronome's API:

```
cat /dev/local/metronome-help
```

You should see:

```
Metronome Resource Manager (ResMgr)

 Usage: metronome <bpm> <ts-top> <ts-bottom>

 API:
  pause [1-9]                        - pause the metronome for 1-9 seconds
  quit                               - quit the metronome
  set <bpm> <ts-top> <ts-bottom>     - set the metronome to <bpm> ts-top/ts-bottom
  start                              - start the metronome from stopped state
  stop                               - stop the metronome; use 'start' to resume
```

echo pause [1-9] > /dev/local/metronome

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 <u>not</u> the next measure, which is |1). For example:

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

echo quit > /dev/local/metronome

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.

echo set <bpm> <ts-top> <ts-bottom> > /dev/local/metronome

When a client writes **set <bpm> <ts-top> <ts-bottom>** to the metronome device (/dev/local/metronome), the metronome resmgr sets the metronome to have <bpm> beats-per-minute with a time signature of <ts-top> over <ts-bottom>. You can assume <bpm>, <ts-top> and <ts-bottom> are integers; you cannot assume the three (3) integers are valid <bpm>, <ts-top> and <ts-bottom>. For example, echo set -3 -2 -1 > /dev/local/metronome, are valid integer values but are not valid values for a metronome. The metronome continues to run on invalid *echo set* integer values; use the last known valid settings before *echo set* was received by the metronome resmgr.

When you enter the following command to set the metronome:

```
echo set 200 5 4 > /dev/local/metronome
```

You should see the metronome's behaviour change to: 200 bpm with a time signature of 5/4 time. If the metronome is currently running, then the metronome's run-time behaviour will change just after writing to the device (i.e. just after you tap the 'enter' key). If the metronome is stopped, then the metronome will run with its new settings when the metronome resmgr receives *echo start > /dev/local/metronome*

echo start > /dev/local/metronome

When a client writes **start** to the metronome device (/dev/local/metronome), the metronome resmgr starts the metronome, presumably from a stopped state. Use the API's 'stop' to stop the running metronome of the resmgr.

```
echo start > /dev/local/metronome
```

Verify the metronome is running and the metronome resmgr is still running as a process:

```
pidin | grep metronome
```

You should see one (1) metronome resmgr process when you run the above command.

echo stop > /dev/local/metronome

When a client writes **stop** to the metronome device (/dev/local/metronome), the metronome resmgr stops the running metronome. Use the API's 'start' to resume the metronome of the resmgr.

```
echo stop > /dev/local/metronome
```

Verify the metronome is no longer running but the metronome resmgr is still running as a process:

```
pidin | grep metronome
```

You should see one (1) metronome resmgr process when you run the above command.

API Error Handling

The metronome resource manager only supports the above input commands: read (i.e. cat command), 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 (i.e. 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 (stderr):

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

Important: the metronome continues to run on bogus input

Acceptance Test Script

You (and your partner if appropriate) will create an acceptance test script that demonstrates the run-time behaviour of your metronome. Make a ksh script named: **acceptance-test.sh**

Recall in Lab05, I provided the unit-test script for you to test the run-time behaviour of your calc_cliient and calc_server.

Run your acceptance test on Neutrino (and not Momentics IDE) and cover the following test 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.
Let your metronome run for 3 to 5 seconds so we can observe the metronome's run-time behaviour.
c) **cat /dev/local/metronome**
Expected: [metronome: 120 beats/min, time signature 2/4, secs-per-interval: 0.25, nanoSecs: 250000000]
d) **cat /dev/local/metronome-help**
Expected: information regarding the metronome resmgr's API, as seen above.
e) **echo set 100 2 4 > /dev/local/metronome**
Expected: metronome regmgr changes settings to: 100 bpm in 2/4 time; run-time behaviour of metronome changes to 100 bpm in 2/4 time.
Let your metronome run for 3 to 5 seconds so we can observer the metronome's run-time behaviour.
f) **cat /dev/local/metronome**
Expected: [metronome: 100 beats/min, time signature 2/4, secs-per- interval: 0.30, nanoSecs: 300000000]
g) **echo set 200 5 4 > /dev/local/metronome**
Expected: metronome regmgr changes settings to: 200 bpm in 5/4 time; run-time behaviour of metronome changes to 200 bpm in 5/4 time.
Let your metronome run for 3 to 5 seconds so we can observer the metronome's run-time behaviour.
h) **cat /dev/local/metronome**
Expected: [metronome: 200 beats/min, time signature 5/4, secs-per- interval: 0.15, nanoSecs: 150000000]
i) **echo stop > /dev/local/metronome**
Expected: metronome stops running; metronome resmgr is still running as a process: pidin | grep metronome.
j) **echo start > /dev/local/metronome**
Expected: metronome starts running again at 200 bpm in 5/4 time, which is the last setting; metronome resmgr is still running as a process: pidin | grep metronome
Let your metronome run for 3 to 5 seconds so we can observer the metronome's run-time behaviour.
k) **cat /dev/local/metronome**
Expected: [metronome: 200 beats/min, time signature 5/4, secs-per- interval: 0.15, nanoSecs: 150000000]
l) **echo stop > /dev/local/metronome**
Expected: metronome stops running; metronome resmgr is still running as a process: pidin | grep metronome.
m) **echo stop > /dev/local/metronome**
Expected: metronome remains stopped; metronome resmgr is still running as a process: pidin | grep metronome.
n) **echo start > /dev/local/metronome**
Expected: metronome starts running again at 200 bpm in 5/4 time, which is the last setting; metronome resmgr is still running as a process: pidin | grep metronome
Let your metronome run for 3 to 5 seconds so we can observer the metronome's run-time behaviour.

o) **echo start > /dev/local/metronome**
Expected: metronome is still running again at 200 bpm in 5/4 time, which is the last setting; metronome resmgr is still running as a process: pidin | grep metronome
Let your metronome run for 3 to 5 seconds so we can observer the metronome's run-time behaviour.

p) **cat /dev/local/metronome**
Expected: [metronome: 200 beats/min, time signature 5/4, secs-per- interval: 0.15, nanoSecs: 150000000]

q) **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.

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

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

t) **echo set 120 2 4 > /dev/local/metronome**
Expected: 1 measure per second. I will use this unit-test to verify the correct cadence of your metronome.
Let your metronome run for 3 to 5 seconds so we can observe the metronome's run-time behaviour.

u) **cat /dev/local/metronome**
Expected: [metronome: 120 beats/min, time signature 2/4, secs-per-interval: 0.25, nanoSecs: 250000000]

v) **cat /dev/local/metronome-help**
Expected: information regarding the metronome resmgr's API, as seen above.

w) **echo Writes-Not-Allowed > /dev/local/metronome-help**
Expected: properly formatted error message, and metronome continues to run.

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

Didn't implement *echo set <bpm> <ts-top> <ts-bottom>*? Then you'll need to quit the metronome each time and then start the metronome with the settings for the test scenario:

echo quit > /dev/local/metronome && metronome <bpm> <ts-top> <ts-bottom>

In-Lab Demonstrations over Zoom

You (and your partner if appropriate) will demonstrate the behaviour of your metronome and run the acceptance test during your scheduled lab period of Week 14.

It's OK if your lab partner is in another Lab Section. If that's the case, pick a scheduled lab period that both of you can attend. If that's not the case due to conflict, then both students will be required to demo separately during their own scheduled lab period.

Deliverables

Before making the zip-file deliverable (next item), please action:

- Format your source code to make it easier for me to read. Momentics IDE will do this for you: Source → Format
- Verify your projects build cleanly. Please action: a) Project → Clean… and b) Project → Build All

Prepare a zip-file that contains the following items:

1) Export your Momentics IDE project as a zip-archive file.
2) 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 collaborated with a partner, list both names (yours + partners) and 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.}

3) Acceptance test script.
4) Name your zip-file according to your Algonquin College username:

   cst8244_assign2_yourUsername{_yourPartnerUsername}.zip

   For Example (partners), cst8244_assign2_bond0007_jaws0001.zip

   For Example (solo), cst8244_assign2_bond0007.zip

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

   - Collaborated with a partner(s)? Two submissions please; each partner is to submit their own zip-file.

Marking Scheme

This ultimate (as in last) assignment is worth 50 marks.

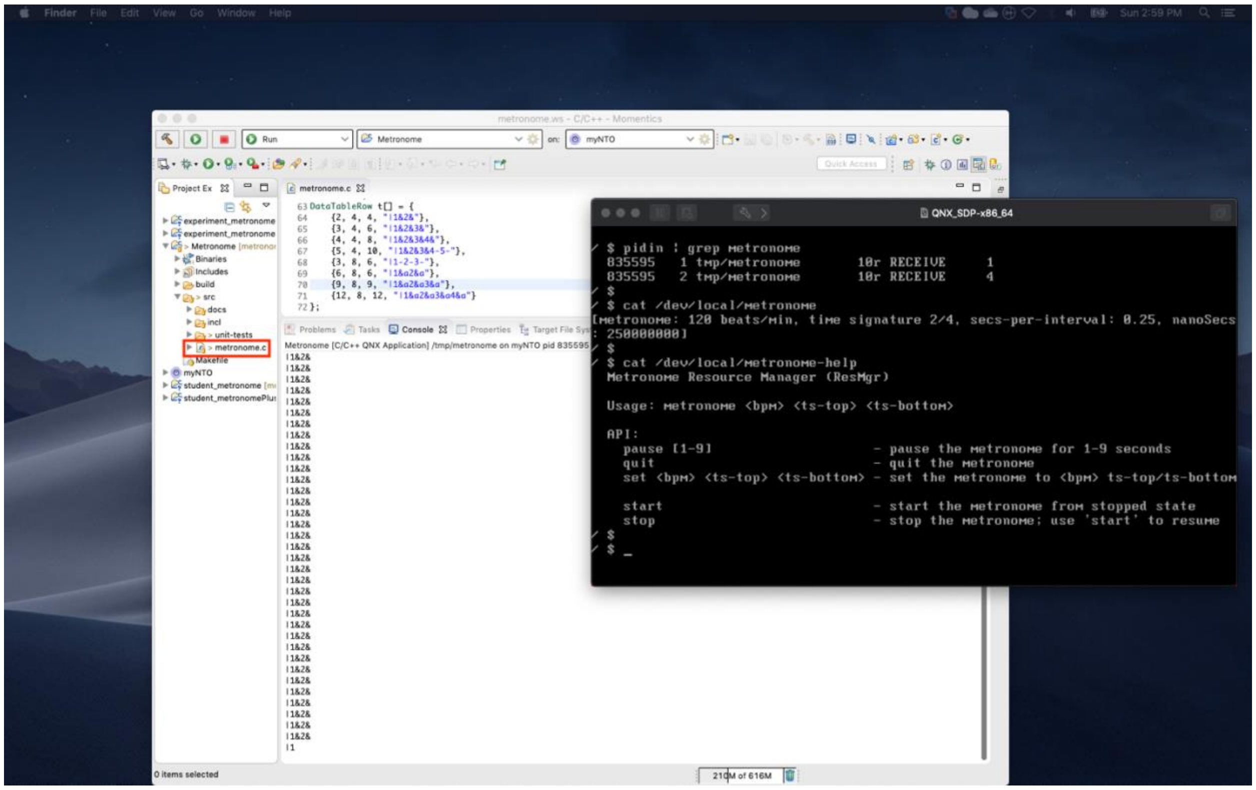
- 20 marks for metronome behaviour: set, start, stop
- 5 marks for the metronome-help device (cat /dev/local/metronome-help)
- 5 marks for current state and status of the metronome device (cat /dev/local/metronome)
- 5 marks for the acceptance test script
- 5 marks for pause <int>
- 5 marks for quit
- 5 marks for API error handling and usage message

Your grade will be capped at 5/20 if your metronome's behaviour is incorrect and 1/5 for the ATS. You're eligible for full marks on the remaining items.

No demo? No problem… subtract 25 points out of 50.

Reference Screenshot

Notice… not a run of my acceptance test script. Rather, I'm showing you a composite screenshot



metronome.c

```
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"}
};
```

Metronome [C/C++ QNX Application] /tmp/metronome on myNTO pid 835595

```
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1&2&
|1
```

QNX_SDP-x86_64

```
/ $ pidin | grep metronome
  835595   1 tmp/metronome       10r RECEIVE     1
  835595   2 tmp/metronome       10r RECEIVE     4
/ $
/ $ cat /dev/local/metronome
[metronome: 120 beats/min, time signature 2/4, secs-per-interval: 0.25, nanoSecs
: 250000000]
/ $
/ $ cat /dev/local/metronome-help
  Metronome Resource Manager (ResMgr)

  Usage: metronome <bpm> <ts-top> <ts-bottom>

  API:
    pause [1-9]                     - pause the metronome for 1-9 seconds
    quit                            - quit the metronome
    set <bpm> <ts-top> <ts-bottom> - set the metronome to <bpm> ts-top/ts-bottom

    start                           - start the metronome from stopped state
    stop                            - stop the metronome; use 'start' to resume
/ $
/ $ _
```