

CST8244 – Real-Time Programming

Lab 7 – Yet Another Simple Resource Manager (YANResMgr)

Introduction

You will be building two programs: **myController** and **myDevice**. The programs will run independently but synchronize by having the **myDevice** program install a resource manager (aka device driver) within QNX Neutrino.

The device status will then be visible to the **myController** by actively reading directly from the device, or by receiving alert *pulses* from the device.

Running the Programs

Startup the device: **# myDevice &**

Startup the controller: **# myController &**

Then you will be able to test your programs using a command scripts containing commands such as:

```
echo status value      > /dev/local/mydevice
echo alert 1           > /dev/local/mydevice
echo status closed     > /dev/local/mydevice
echo alert 2           > /dev/local/mydevice
```

The device should operate as follows:

1. update its internal status buffer with: **value**. Where *value* is a string of characters following the **status <space>** prefix. Please note a blank space separates **status** and *value*. Allow for up to 255 characters in the **value**. This **value** will then be available to the controller if the device is read (e.g., using `fscanf(...)`).
2. send an “alert”, in the form of a pulse to the controller, to notify that the **alert <small_int>** event has been written to the device. From above, I use the **echo** command to write to the device, **/dev/local/mydevice** (FQN – fully qualified name). The value of <small_int> must be in the range 1 to 99, inclusive. You can assume integer values are sent; you cannot assume the integer value is in the range of 1 – 99 (inclusive). If the integer value is outside this range, print a warning message (“Integer value is not between 1 and 99 (inclusive)”), but do *not* terminate --- the device program is to continue running (i.e. its fault tolerant, well a little at least).

Upon startup, the **myController** program should read the status of the device, and then whenever a pulse is received, the **myController** program should output the value of the integer sent with the pulse, read the status of the device again, and output a message to the console with the current device status.

Marking Scheme

The lab is marked out of 30 points.

- 10 marks for **myController**
 - Must gracefully terminate (call `name_detach()` & exit with success) on: **status closed** followed by **alert <small_int>**
- 10 marks for **myDevice**
 - Domain range check of alert's `<small_int>`
- 10 marks for demonstration by screen-shot(s)

Marks will be deducted for not checking return codes for errors or not properly freeing any dynamic memory you allocate. Comments are not necessary, but are always welcome.

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. Make a set of screen-shots showing the following scenario:

```
echo status value      > /dev/local/mydevice
echo alert 1           > /dev/local/mydevice
echo status closed     > /dev/local/mydevice
echo alert 2           > /dev/local/mydevice
```

Additionally, your screenshot must show that the controller gracefully terminated.

2. Export your Momentics IDE projects as a zip-archive file.
3. A "README.txt" file reporting the status of your lab. Follow this template:

Title {give your work a title}

Author

{Please sign your work. For example: @author Gerald.Hurdle@AlgonquinCollege.com}

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.}

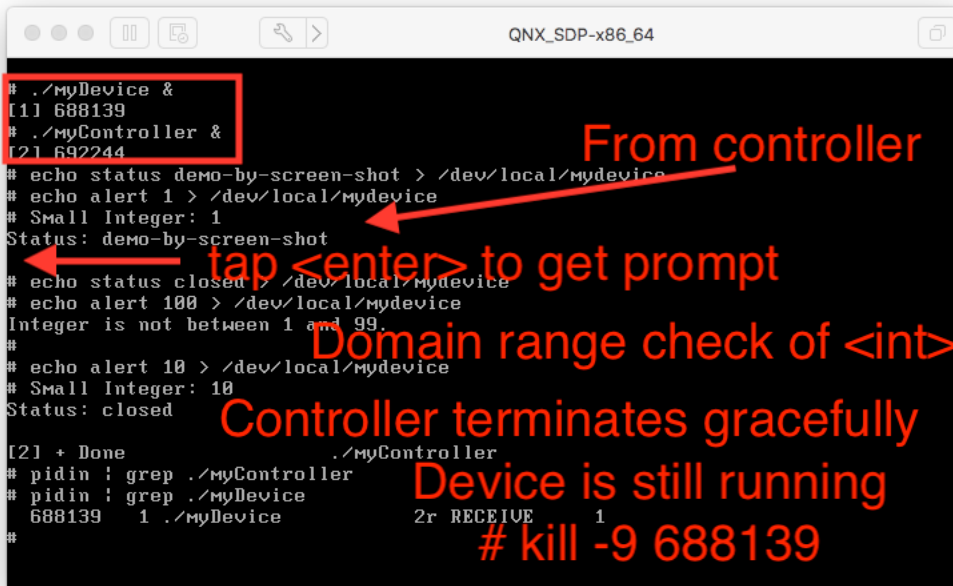
4. Name your zip-file according to your Algonquin College username:
cst8244_lab7_yourUsername.zip

For Example, cst8244_lab7_bond0007.zip

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

Reference Screenshot

Compare your screenshot to mine:



```
# ./myDevice &
[1] 688139
# ./myController &
[2] 692244

# echo status demo-by-screen-shot > /dev/local/mydevice
# echo alert 1 > /dev/local/mydevice
# Small Integer: 1
Status: demo-by-screen-shot
# echo status closed > /dev/local/mydevice
# echo alert 100 > /dev/local/mydevice
Integer is not between 1 and 99.
#
# echo alert 10 > /dev/local/mydevice
# Small Integer: 10
Status: closed

[2] + Done ./myController
# pidin : grep ./myController
# pidin : grep ./myDevice
688139 1 ./myDevice 2r RECEIVE 1
#
```

From controller

tap <enter> to get prompt

Domain range check of <int>

Controller terminates gracefully

Device is still running

kill -9 688139