Exercise 10: QNX Resource Manager

In the previous exercise, we performed inter-process communication using the message passing primitives in QNX. But we encountered an issue with how to announce the server's existence, where we had to resort to using the file system to know the relevant process ID and channel ID.

The more elegant and idiomatic solution to this problem is to use Resource Managers, which announce their names in the /dev folder, and only have a single channel where the send and receive operations are performed with standard input-output operations like read and write. While the underlying mechanisms are still the channel and message primitives, we will no longer be using these directly.

You can read more about Resource Managers here:
http://www.qnx.com/developers/docs/6.5.0_sp1/topic/com.qnx.doc.neutrino_resmgr/overview.html

1. Resource Managers

A Resource Manager is represented as a file in the /dev folder, and a client communicates with it by opening, reading form, and writing to this file. Creating a Resource Manager program requires a bit of boilerplate code, so to get you started a basic Resource Manager program is provided on Blackboard.

This program creates a resource called "myresource" (found in /dev/myresource), and implements functionality for reading only. All other file operations are automatically set to their do-nothing defaults, so to override calls to operations like open and write, you will have to first create a your own function that is called when a client calls that file operation, then attach that function to the connect_funcs or io_funcs data structures. These data structures contain your overrides for the available file operations, and are passed to the call to `resmgr_attach`, which makes this Resource Manager available in the file system. Then, the call to `dispatch_block` waits for a client to call some file operation, and the call to `dispatch_handler` chooses and runs the corresponding function from the overrides you provided.

Task A:

Compile and run this resource manager. Run `cat /dev/myresource` to perform a read operation on the resource, and verify that it works (just use a terminal window in the QNX virtual machine directly).

Task B:

Modify the resource such that you can both read and write to it.

- The resource should store a string of characters that can be retrieved later.
- Running `echo "this is some text" > /dev/myresource` should overwrite the currently stored text.
- Running `cat /dev/myresource` should retrieve the text.

More information about handling read and write messages can be found here:
http://www.qnx.com/developers/docs/6.5.0_sp1/topic/com.qnx.doc.neutrino_resmgr/read_write.html
Specifically the sample code for handing write calls:
http://www.qnx.com/developers/docs/6.5.0_sp1/topic/com.qnx.doc.neutrino_resmgr/read_write.html#Sample_IO_WRITE

The sample code contains functionality that isn't strictly necessary. Start simple, and just create the `io_write` function and have it call `printf` and return - and remember to hook it in to the `io_funcs` data structure in `main`.

You do not need to care about the testing of `iofunc_write_verify`, the testing of `xtype`, or the change to `ocb->attr->flags`. You can also define a large global char array, instead of allocating it with `malloc`.

The things that you *do* need to do are:

- Set the number of bytes that the client's write command should return
- Call `resmgr_msgread` to get the written data
- Return `_RESMGR_NPARTS(0)`.

Task C:

Create a counting resource, which keeps counting in the direction requested.

- Reading the resource (with cat) should return the current value of the resource.
- The resource should accept three different commands (write operations):
  - "up", "down", and "stop"
- The counter should increment/decrement every 100ms. You will have to spawn a separate thread to perform the counting itself.

2. Qnet, and resources on other nodes

A Resource Manager can be accessed not only by processes on the local machine, but also on other machines connected over the network. To enable Qnet, we just have to create a file in the global system configuration:

```
touch /etc/system/config/useqnet
```

Then, we create another QNX machine in VirtualBox, by right-clicking our target while it is not running, and selecting "Clone". You will then get an identical computer, so to make the two machines distinct we will have to change one of the machine's MAC address. Open the settings for one of the machines, go to the "Network" menu, expand the "Advanced" dropdown menu, and hit the refresh symbol next to the MAC address to generate a new one.

Start both the machines. We have to give a different host name to (at least) one of them, so go to "Configure > Network" in the right hand pane, type in a different host name, hit Apply (this may take a while), and reboot. When you have rebooted, confirm that both machines have different IP addresses. If you then type

```
ls /net
```

you should see both host names show up from both machines. From this folder, you can access the file system of one machine from the other, which means you can access the Resource Managers from the other machine too.

Task D:

Run the counting resource manager from one machine, then access it with `cat` and `echo` from both.