Semaphores

Part A

Here's my output when running ex03a

```
xenomai@ieu:~/excercises/semaphores$ ./ex03a
I am taskOne and global = 1.....
I am taskOne and global = 2.....
I am taskOne and global = 3.....
I am taskOne and global = 4.....
I am taskOne and global = 5.....
I am taskOne and global = 6.....
I am taskOne and global = 7.....
I am taskOne and global = 8.....
I am taskOne and global = 9.....
I am taskOne and global = 10.....
I am taskTwo and global = 9----------
I am taskTwo and global = 8----------
I am taskTwo and global = 7----------
I am taskTwo and global = 6----------
I am taskTwo and global = 5----------
I am taskTwo and global = 4----------
I am taskTwo and global = 3----------
I am taskTwo and global = 2----------
I am taskTwo and global = 1----------
I am taskTwo and global = 0----------
```

Task 1 is started first and has same priority as task 2. Task 1 will run to completion and then task 2 will begin to execute. This causes the global variable to increment to 10 then back to 0.

Part B

Here's my output:

```
xenomai@ieu:~/excercises/semaphores$ ./ex03b
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
```

Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0
Task One: Global = 1
Task Two: Global = 0

Essentially, I am using semaphores to ping-pong between task 1 and 2. Both wait for a signal (rt_sem_p) and both end with signaling (rt_sem_v). The semaphore counter only toggles between 0 and 1 in this case and is the reason why rt_sem_p at the beginning of the task waits until rt_sem_v at the end of the task. rt_sem_p decreases the counter unless it is already 0. If it is 0, rt_sem_p waits until the counter is incremented by a rt_sem_v.

Since the tasks are the same priority, task execution will ping-pong using semaphores. The entire process is started by rt_sem_v which incremements the semaphore counter to above 0 allowing task 1 to begin executing.

Part C

RT_SEM_V Solution

My solution has each task have its own semaphore. I signal the semaphore for task 5 after all tasks have been started. Task 5 runs and signals task 4's semaphore once its done. This then repeats for all tasks. Even though each have the same priority, I have made it so that task 5 has the highest and task 1 has the lowest.

Not the nicest looking solution, but I did learn that you can make a semaphore variable an array and index it. This works as long as you create each semaphore in the array with a unique string name (hence the sprintf statement).