Schiffer, Nickolas

CMPE 146

Producer_Consumer_Tasks

1. *Print a message **before** and **after** sending the orientation to the queue*

```
producer: sending orientation 'right'
consumer: receiving orientation 'right'
producer: sent orientation 'right'
```

This behaves as expected, the producer sends an item to the queue which wakes up the consumer (higher priority). The consumer runs and then sleeps which causes producer to resume and print the 'sent' message.

*Print a message **after** the `consumer task` receives an item from the queue*
Use the same priority for both tasks, and note down the order of the print-outs

In this case, the tasks switch between each other in conventional round robin fashion. This causes the order of the statements to change accordingly.

```
consumer: receiving orientation 'right'
producer: sending orientation 'right'
producer: sent orientation 'right'

consumer: receiving orientation 'right'
producer: sending orientation 'right'
producer: sent orientation 'right'
```

Use higher priority for the receiving task (`consumer task`), and note down the order of the print-outs.

When the producer enqueues a direction, the consumer is resumed as it is waiting on receiving from the queue. Due to the consumer being a higher priority task, it will immediately run. It is only when the consumer sleeps that the producer will run again.

```
producer: sending orientation 'right'
consumer: receiving orientation 'right'
producer: sent orientation 'right'

producer: sending orientation 'right'
consumer: receiving orientation 'right'
producer: sent orientation 'right'
```

2. *What if you use ZERO block time while sending an item to the queue, will that make any difference?*
   The block time is only used if the queue is full. With this setup, it will not make a difference because the queue is being dequeued at the same rate as is it being populated.
3. *What is the purpose of the block time during `xQueueReceive()` ?*

The block time determines how long the consumer will block and wait for an object to be queued. In my program, the queue will wait indefinitely.

main.cpp

```
/*
 * Nickolas Schiffer CMPE 146
 * Producer Consumer Tasks
 */

#include <command_handler.hpp>
#include <FreeRTOSConfig.h>
#include <queue.h>
#include <scheduler_task.hpp>
#include <stdio.h>
#include <str.hpp>
#include <task.h>
#include <tasks.hpp>
#include <acceleration_sensor.hpp>



QueueHandle_t q = NULL;

TaskHandle_t vProducerHandle;

Acceleration_Sensor orientation_sens = Acceleration_Sensor::getInstance();

typedef enum { invalid, up, down, left, right } orientation_t;


CMD_HANDLER_FUNC(orientationToggleHandler){

    if (cmdParams == "on") {
        vTaskResume(vProducerHandle);
        printf("Orientation monitor toggled on\n");
    }
    else if (cmdParams == "off") {
        vTaskSuspend(vProducerHandle);
        printf("Orientation monitor toggled off\n");
    }
    else
        return false;

    return true; /* return true if command was successful */
}

void vProducer(void *pvParameters) /* LOW priority */
{
  orientation_t orientation = invalid;
  int16_t x, y;
  vTaskSuspend(vProducerHandle);
  while (1) {
    x = orientation_sens.getX();
    y = orientation_sens.getY();


    if (abs(x) > abs(y)){      //left/right
        if (x > 0)
            orientation = left;
```

```
        else if (x < 0)
            orientation = right;
        else
            orientation = invalid;
    }
    else if (abs(x) < abs(y)){ //up/down
        if (y > 0)
            orientation = up;
        else if (y < 0)
            orientation = down;
        else
            orientation = invalid;
    }
    else
        orientation = invalid;

    switch (orientation){
        case 0:
            printf("producer: sending orientation 'invalid'\n");
            break;
        case 1:
            printf("producer: sending orientation 'up'\n");
            break;
        case 2:
            printf("producer: sending orientation 'down'\n");
            break;
        case 3:
            printf("producer: sending orientation 'left'\n");
            break;
        case 4:
            printf("producer: sending orientation 'right'\n");
            break;
    }
    xQueueSend(q, &orientation, 0);
    switch (orientation){
        case 0:
            printf("producer: sent orientation 'invalid'\n\n");
            break;
        case 1:
            printf("producer: sent orientation 'up'\n\n");
            break;
        case 2:
            printf("producer: sent orientation 'down'\n\n");
            break;
        case 3:
            printf("producer: sent orientation 'left'\n\n");
            break;
        case 4:
            printf("producer: sent orientation 'right'\n\n");
            break;
    }
    vTaskDelay(100);
  }
}

void vConsumer(void *pvParameters) /* HIGH priority */
{
  orientation_t orientation;
  while (1) {
    xQueueReceive(q, &orientation, portMAX_DELAY);
    switch (orientation){
```

```
        case 0:
            printf("consumer: receiving orientation 'invalid'\n");
            break;
        case 1:
            printf("consumer: receiving orientation 'up'\n");
            break;
        case 2:
            printf("consumer: receiving orientation 'down'\n");
            break;
        case 3:
            printf("consumer: receiving orientation 'left'\n");
            break;
        case 4:
            printf("consumer: receiving orientation 'right'\n");
            break;
    }
  }
}

int main(void)
{


  // Queue handle is not valid until you create it
  scheduler_add_task(new terminalTask(PRIORITY_HIGH));
  q = xQueueCreate(10, sizeof(orientation_t));
  xTaskCreate(vConsumer, "Consumer", 1024, NULL, PRIORITY_HIGH, NULL);
  xTaskCreate(vProducer, "Producer", 1024, NULL, PRIORITY_LOW, &vProducerHandle);
  scheduler_start();
  return -1;

}
```