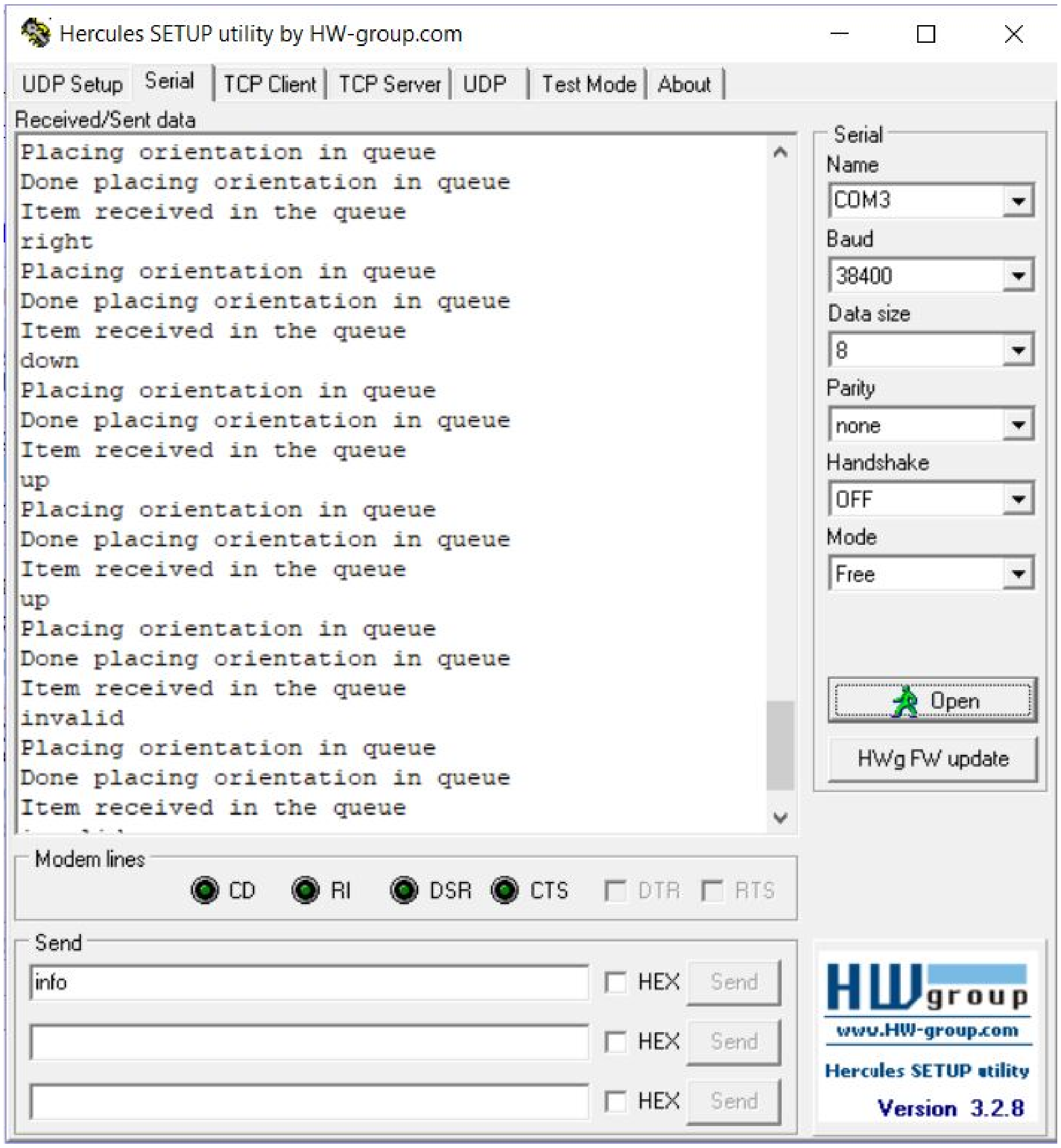
main.cpp

```
#include "CP.hpp"
#include <FreeRTOS.h>
#include <stdint.h>
#include <Task.h>
#include <Tasks.hpp>
#include "queue.h"
#include "acceleration_sensor.hpp"
#include "printf_lib.h"
#include "command_handler.hpp"

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

int16_t x = 0;
int16_t y = 0;

Acceleration_Sensor acc = Acceleration_Sensor::getInstance();       //getting
an object from Singleton class
QueueHandle_t qu;
//queue_handle

orientation_t accval(void)                //Function to decide the orientation
of the accelerometer
{
    orientation_t ab;                   //variable to be returned as the
orientation
    if (y > 300 && y < 1000)
        ab = right;
    else if (y < -300 && y > -1000)
        ab = left;
    else if (x > 300 && x <1000)
        ab = up;
    else if (x < -300 && x > -1000)
        ab = down;
    else
        ab = invalid;
    return ab;
}
void producer(void *p)
{
    int h;
  while (1) {
     x = acc.getX();    //get the value of x-axis from the function acc
     y = acc.getY();    //get the value of y-axis from the function acc
     h = accval();      //get the current orientation
     u0_dbg_printf("Placing orientation in queue\n");
    xQueueSend(qu, &h, 10);     //Place the orientation in the queue
    u0_dbg_printf("Done placing orientation in queue\n");
    vTaskDelay(1000);
  }
}
void consumer(void *p)
{
  int16_t a = 0;
  while (1) {
    if(xQueueReceive(qu, &a, portMAX_DELAY)) {
        u0_dbg_printf("Item received in the queue\n");
        switch (a) {        //switch case for printing out the orientation
from the queue
            case up:
                u0_dbg_printf("up\n");
                break;
            case down:
                u0_dbg_printf("down\n");
                break;
            case left:
                u0_dbg_printf("left\n");
                break;
            case right:
                u0_dbg_printf("right\n");
                break;
            case invalid:
                u0_dbg_printf("invalid\n");
                break;
            default:
                u0_dbg_printf("INVALID\n");
            }
    }
     else
         u0_dbg_printf("Did not receive orientation from queue");

    vTaskDelay(1000);
  }
}
int main(void)
{
  qu = xQueueCreate(10, sizeof(int));       //Queue size of 10 with each slot
having a size of integer
  xTaskCreate(consumer, (const char*) "Consumer", 512, NULL, 2, NULL);
  xTaskCreate(producer, (const char*) "Producer", 512, NULL, 1, NULL);
  vTaskStartScheduler();
  return 0;
}
```

Terminal Output

a) Consumer Task priority: 1, Producer Task priority: 1



Hercules SETUP utility by HW-group.com

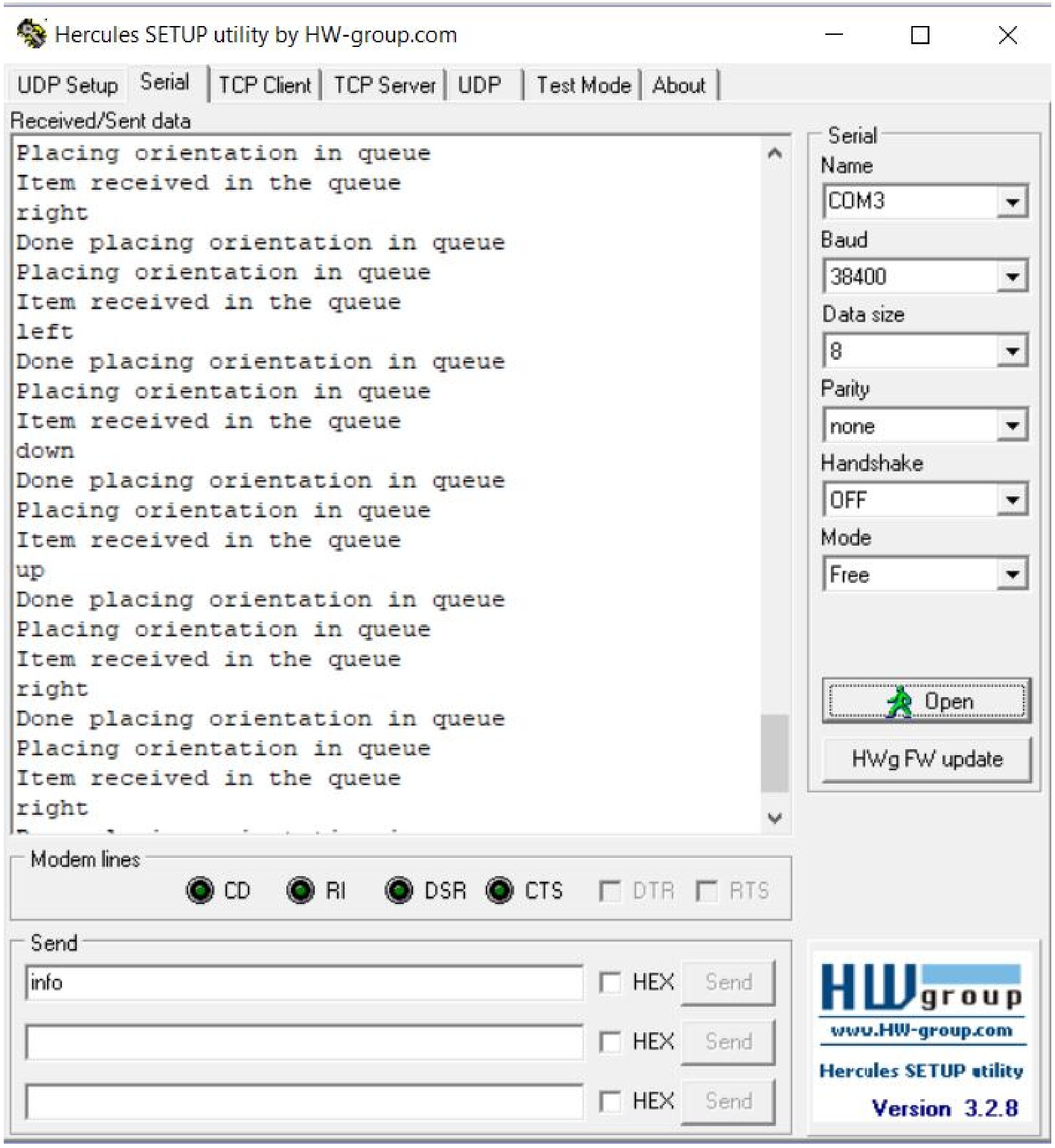
Serial: Name COM3, Baud 38400, Data size 8, Parity none, Handshake OFF, Mode Free

```
Placing orientation in queue
Done placing orientation in queue
Item received in the queue
right
Placing orientation in queue
Done placing orientation in queue
Item received in the queue
down
Placing orientation in queue
Done placing orientation in queue
Item received in the queue
up
Placing orientation in queue
Done placing orientation in queue
Item received in the queue
up
Placing orientation in queue
Done placing orientation in queue
Item received in the queue
invalid
Placing orientation in queue
Done placing orientation in queue
Item received in the queue
```

Send: info

Here the priority of the consumer and the producer task is the same and hence it will execute the task in round robin. As we can see in the output the data is travelling from one task to the other in order i.e. first the orientation is ready to be placed on the queue, then the task is done placing it on the queue and in the consumer task it is received, and later orientation is displayed.

b) Consumer Task priority: 2, Producer Task priority: 1



```
Placing orientation in queue
Item received in the queue
right
Done placing orientation in queue
Placing orientation in queue
Item received in the queue
left
Done placing orientation in queue
Placing orientation in queue
Item received in the queue
down
Done placing orientation in queue
Placing orientation in queue
Item received in the queue
up
Done placing orientation in queue
Placing orientation in queue
Item received in the queue
right
Done placing orientation in queue
Placing orientation in queue
Item received in the queue
right
```

Serial
Name: COM3
Baud: 38400
Data size: 8
Parity: none
Handshake: OFF
Mode: Free

Send: info

Here the priority of the consumer task is higher than the producer task. The higher priority task runs first and then goes into the blocked state as the queue is empty thus allowing the producer task to run. Thus, as soon as the item is placed on the queue the consumer task becomes ready and reads the orientation from the queue and goes into sleep because of vTaskDelay(). Then the producer task resumes and states that it is done placing the item on the queue following the rule, higher priority ready task always runs.

Additional questions and their answers:

a) What if you use ZERO block time while sending an item to the queue, will that make any difference?

The block time for sending an item in the queue is the time for which a task will remain in blocked state to wait for the queue to get empty in case if it is full. So, when the block time for sending an item to the queue is zero, it won't send the task into the blocked state if the queue is full i.e. if the task cannot write in the queue. Here block time being zero the task will fail to write into the queue and the receiving task won't be able to get that data from the queue. On the other hand, if the queue is empty than it won't make a difference even if the block time is zero, the task will simply write into the queue. So, zero block time will only make a difference when the queue is full.

b) What is the purpose of the block time during xQueueReceive() ?

The purpose of the block time during xQueueReceive() is to send the task into the blocked state for that amount of time only if the queue is empty and thereby wait for an item to be received into the queue. The task only remains in the blocked state for that amount of time and then even if the queue is still empty it will move the task into the ready state.