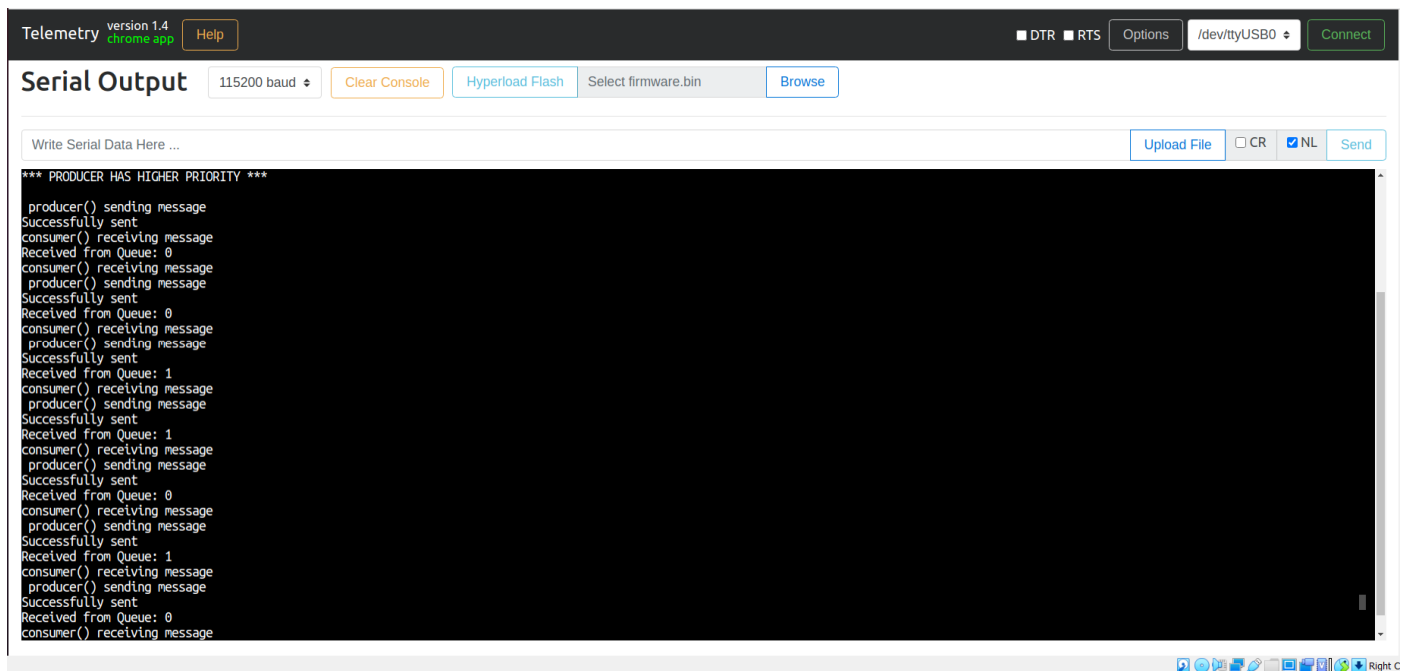


LAB 8: PRODUCER AND CONSUMER

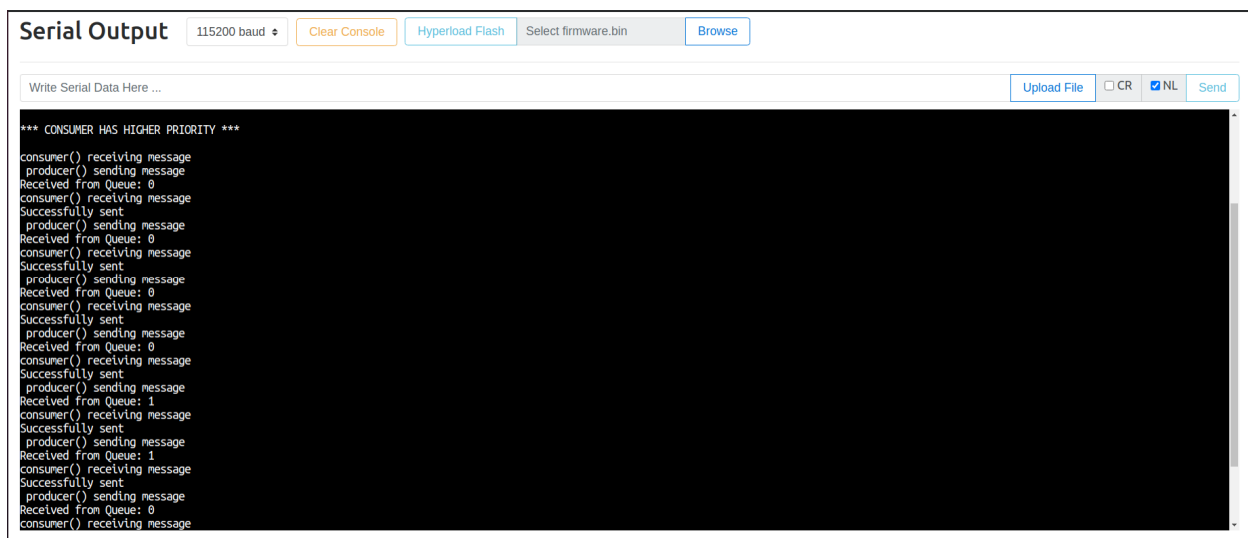
Case 1: Producer has higher Priority

When the producer is set to a higher priority the producer is executed first and prints the first print statement. After that it writes into the queue and then prints the second statement. Since the consumer has low priority it is executed next. The next statements of printing and reading from the queue happens sequentially. There is no task going to sleep since the producer has higher priority due to which the writing and reading from queue happens sequentially. This cycle repeats infinitely. Whenever the button is pressed a 1 written as data in the queue else a 0 is written. The queue is created such that only one data can be written into the queue: `xQueueCreate(1, sizeof(switch_e))`.



Case 2: Consumer has higher Priority

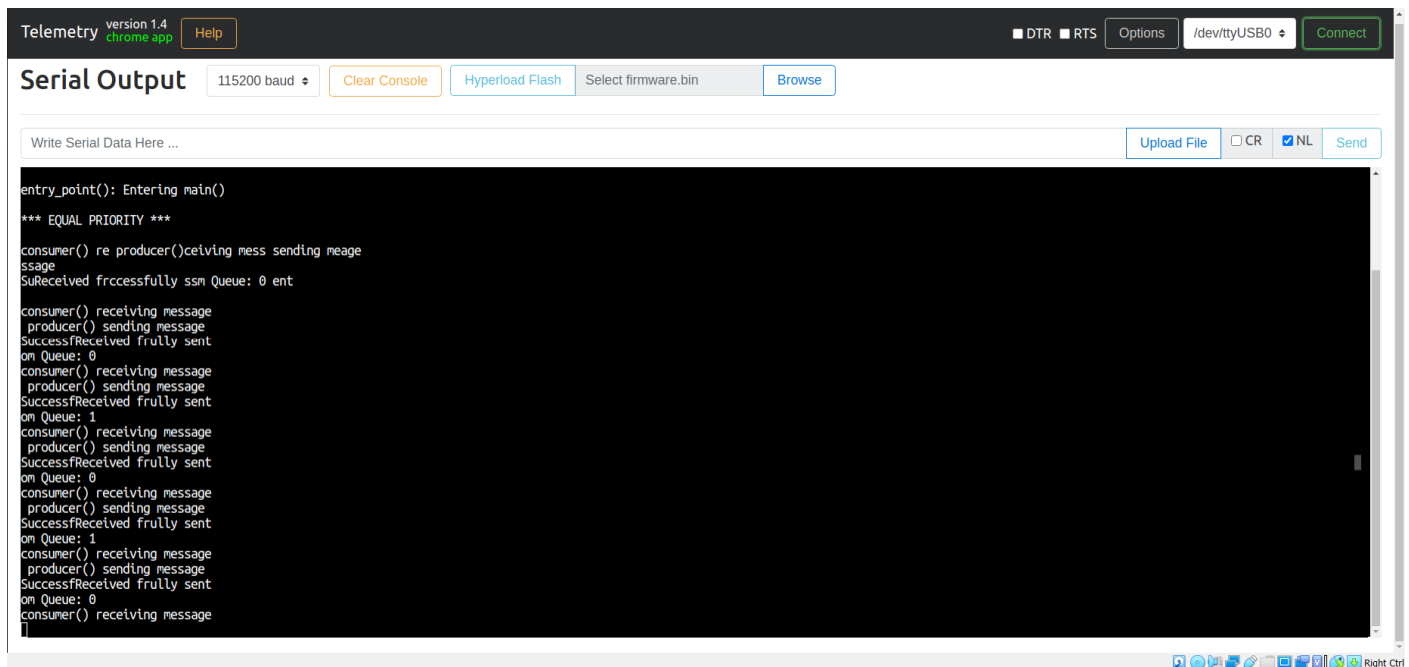
When the consumer is set to a higher priority the consumer is executed first and prints the first print statement. When it tries to execute the next statement `xqueuereceive()` statement it realizes that the queue is empty and hence FreeRTOS sleeps this task and goes to execute the producer task and prints the first statement. Producer then sends data into queue and that moment the consumer task is woken up by context switching and that's why it interrupts the producer task and writes the value read from queue. Once again since consumer has higher priority it will execute the consumer task first and then realizing the queue is empty it sleeps the task. Since the second print statement was not executed in the last cycle due to context switching, that statement will execute once again and then the producer task is executed to write into the queue. This cycle repeats infinitely. Whenever the button is pressed a 1 written as data in the queue else a 0 is written. The queue is created such that only one data can be written into the queue: `xQueueCreate(1, sizeof(switch_e))`.



```
Serial Output 115200 baud Clear Console Hyperload Flash Select firmware.bin Browse
Write Serial Data Here ... Upload File CR NL Send
*** CONSUMER HAS HIGHER PRIORITY ***
consumer() receiving message
producer() sending message
Received from Queue: 0
consumer() receiving message
Successfully sent
producer() sending message
Received from Queue: 0
consumer() receiving message
Successfully sent
producer() sending message
Received from Queue: 0
consumer() receiving message
Successfully sent
producer() sending message
Received from Queue: 0
consumer() receiving message
Successfully sent
producer() sending message
Received from Queue: 1
consumer() receiving message
Successfully sent
producer() sending message
Received from Queue: 1
consumer() receiving message
Successfully sent
producer() sending message
Received from Queue: 0
consumer() receiving message
```

Case 3: Both have equal Priority

When both the tasks have equal priority there is a constant context switching that occurs which leads to interference when writing the print statements. Since both have equal priority any task can be executed first. In this case the consumer is executed first, and it realizes that the queue is empty and hence goes to sleep. After this the producer task is executed and prints its first statement and writes the data into the queue. As soon as the data is written the consumer task is woken up and while the producer is printing the second print statement the consumer also comes in and wants to print its second print statement. This is the reason that both the tasks second print statements are clashing with one another while the first print statements are written without any interference. Whenever a switch is pressed 1 is written on the queue else a 0 is written. This process goes on infinitely.



The screenshot shows the Telemetry version 1.4 chrome app interface. The top bar includes a 'Help' button, a 'DTR' button, an 'RTS' button, an 'Options' button, a '/dev/ttyUSB0' dropdown, and a 'Connect' button. Below the top bar, there is a 'Serial Output' section with a '115200 baud' dropdown, a 'Clear Console' button, a 'Hyperload Flash' button, a 'Select firmware.bin' dropdown, and a 'Browse' button. The main area of the interface is a text input field labeled 'Write Serial Data Here ...' with an 'Upload File' button, a 'CR' checkbox, a 'NL' checkbox, and a 'Send' button. The output window displays the following text:

```
entry_point(): Entering main()
*** EQUAL PRIORITY ***
consumer() re producer()ceiving mess sending meage
ssage
SuReceivd frccessfully ssm Queue: 0 ent
consumer() receiving message
producer() sending message
SuccessReceived frully sent
on Queue: 0
consumer() receiving message
producer() sending message
SuccessReceived frully sent
on Queue: 1
consumer() receiving message
producer() sending message
SuccessReceived frully sent
on Queue: 0
consumer() receiving message
producer() sending message
SuccessReceived frully sent
on Queue: 1
consumer() receiving message
producer() sending message
SuccessReceived frully sent
on Queue: 0
consumer() receiving message
```

ADDITIONAL QUESTIONS:

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

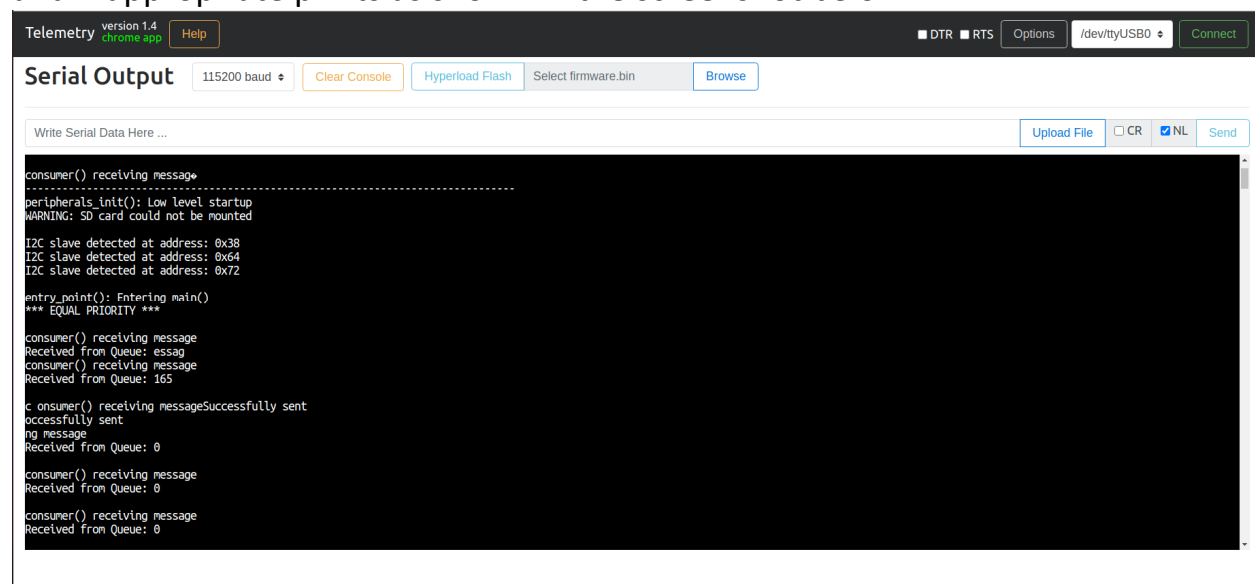
Answer: The structure of xQueueReceive() is as follows:

*xQueueReceive (QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait)*

The third parameter is the block time which is the maximum time a task can wait for the sent data to be received in case the queue is empty. This time is on tick periods which is generally in ms. The current code sets this parameter to **portMAX_DELAY** so that the queue waits for the maximum time if the queue is not received immediately before executing the task.

2. What if you use ZERO block time during xQueueReceive()?

Answer: When the block time is set to 0 the consumer does not wait for the queue value but returns immediately when queue is empty. This leads to incorrect values and inappropriate prints as shown in the screenshot below.



```
Telemetry version 1.4
Help
DTR RTS Options /dev/ttyUSB0 Connect

Serial Output 115200 baud Clear Console Hyperload Flash Select firmware.bin Browse

Write Serial Data Here ... Upload File CR NL Send

consumer() receiving message
peripherals_init(): Low level startup
WARNING: SD card could not be mounted
I2C slave detected at address: 0x38
I2C slave detected at address: 0x64
I2C slave detected at address: 0x72
entry_point(): Entering main()
*** EQUAL PRIORITY ***
consumer() receiving message
Received from Queue: essag
consumer() receiving message
Received from Queue: 165
consumer() receiving messageSuccessfully sent
ccessfully sent
ng message
Received from Queue: 0
consumer() receiving message
Received from Queue: 0
consumer() receiving message
Received from Queue: 0
```

Both the tasks have equal priority and hence a time slicing will occur. Initially the queue read an incorrect value since it did not wait to read from the queue and printed a previous read value. The next instance it reads that there is no button press and hence prints a 0 but it does not give the producer a chance to execute. When the button is pressed it reads a 1 for the amount more than how much the button was pressed for as shown below:

```
Received from Queue: 0
Successfully sent
Queue: 0
consumer() receiving message
Received from Queue: 1
consumer() receiving message
Received from Queue: 1
consumer() receiving message
Received from Queue: 1
consumer() receiving message
Received from Queue: 1
consumer() receiving message
Received from Queue: 1
consumer() receiving message
Received from Queue: 1
consumer() receiving message
Received from Queue: 1
consumer() receiving message
Received from Queue: 1
```

Here when button is pressed the producer print statements are printed and then a 1 is read by the consumer for an amount more than required. This error occurs because the queue receive is not blocked for enough time to read from queue and a change in the task is delayed output when read from queue receive.

EXTRA CREDIT:

This screenshot is at the start of program to show the print of all the available CLI commands:

```
peripherals_init(): Low level startup
WARNING: SD card could not be mounted
I2C slave detected at address: 0x38
I2C slave detected at address: 0x64
I2C slave detected at address: 0x72
entry_point(): Entering main()
*** PRODUCER HAS HIGHER PRIORITY ***

List of commands (use help <name> to get full help if you see ...):
  crash : Deliberately crashes the system to demonstrate how ...
  i2c    : I2C read 0x00 0xRR <n>...
  tasklist : Outputs list of RTOS tasks, CPU and stack usage...
  taskcontrol : taskcontrol suspend <name of the task>...

() sending message
Successfully sent
consumer() receiving message
Received from Queue: 0
consumer() receiving message
producer() sending message
Successfully sent
Received from Queue: 0
consumer() receiving message
producer() sending message
Successfully sent
```

The below screenshot is when the producer task is suspended by typing the command “taskcontrol suspend tx” in the telemetry. The keyword ‘tx’ was used because in my code the second parameter in ‘xtaskcreate()’ corresponds to the name of task and that’s how the CLI command can identify the function and suspend or resume it.

Serial Output

115200 baud ▾ Clear Console Hyperload Flash Select firmware bin Browse

Write Serial Data Here ... Upload File ☐ CR ☒ NL Send

```
.producer() sending message  
Successfully sent  
Received from Queue: 0  
.consumer() receiving message  
.producer() sending message  
Successfully sent  
Received from Queue: 0  
.consumer() receiving message  
.producer() sending message  
Successfully sent  
Received from Queue: 0  
.consumer() receiving message  
.producer() sending message  
Successfully sent  
Received from Queue: 0  
.consumer() receiving message  
.producer() sending message  
Successfully sent  
Received from Queue: 0  
.consumer() receiving message  
.producer() sending message  
Successfully sent  
Received from Queue: 0  
.consumer() receiving message  
taskcontrol suspend tx  
.....
```

After Resuming the producer task (tx): As seen, there is no execution that takes place until the producer is resumed.

Telemetry

version 1.4
chrome app

Help

DTR

RTS

Options

/dev/ttyUSB0

Connect

Serial Output

115200 baud

Clear Console

Hyperload Flash

Select firmware.bin

Browse

Write Serial Data Here ...

Upload File

☐ CR
 ☒ NL

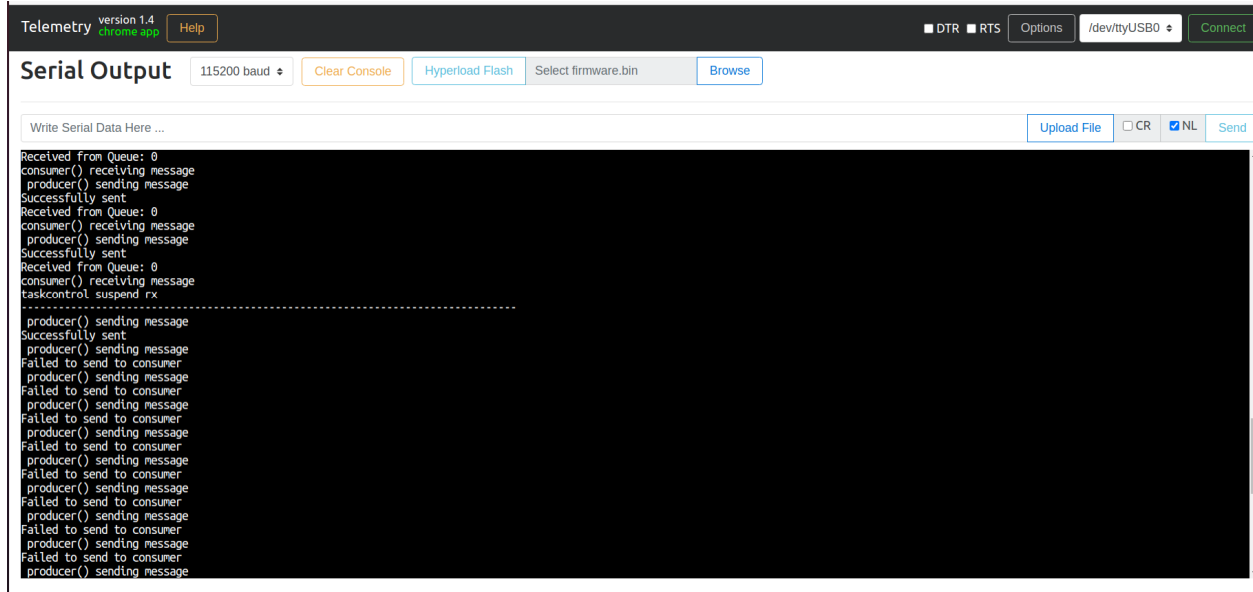
Send

```

producer() sending message
Successfully sent
Received from Queue: 0
consumer() receiving message
producer() sending message
Successfully sent
Received from Queue: 0
consumer() receiving message
producer() sending message
Successfully sent
Received from Queue: 0
consumer() receiving message
taskcontrol suspend tx
-----
taskcontrol resume tx
producer-----
() sending message
Successfully sent
Received from Queue: 0
consumer() receiving message
producer() sending message
Successfully sent
Received from Queue: 0
consumer() receiving message
producer() sending message
Successfully sent
Received from Queue: 0
consumer() receiving message
producer() sending message
Successfully sent
Received from Queue: 0

```

Below screenshot is when consumer(rx) is suspended by typing the command “taskcontrol suspend rx” in telemetry. This leads to execution of only the producer task which is of higher priority.



After consumer (rx) is resumed program resumes normal functionality as shown below:

