|  |  |  |  |
| --- | --- | --- | --- |
| BN | Section | Student ID | اسم الطالب |
| **22** | **4** | **9220917** | **نور أشرف أحمد ماهر** |
| **27** | **4** | **9220954** | **يارا أسامة مهتدى إدريس** |

# System Design

1. Call vResetFunction1 in main function.
2. Create queue, 4 tasks (2 senders +1 higher priority sender +1 receiver) ,create 4 semaphores for synchro
3. Declaration of Function which calculate random period

|  |
| --- |
| int getRandomTimePeriod(int lowerBound, int upperBound) { return (rand() % (upperBound - lowerBound + 1)) + lowerBound;} |

1. Create 1 fixed timer for receiver .

|  |
| --- |
| xReceiverTimer = xTimerCreate("ReceiverTimer", pdMS\_TO\_TICKS(Treceiver), pdTRUE, NULL, vReceiverTimerCallback); |

5.Create 3 timers with random periods.

6. sender task: while (1) { 1. Wait for the semaphore to be released by the timer callback. 2. Create the message. 3.Try to send the message to the queue (counter of sent++ / (if queue is full) blocked ++). 4. Sleep for a random period.

|  |
| --- |
| // create message by sender task  snprintf(message, 20, "Time is %lu", xTaskGetTickCount()); |

7. receiver task : while(1){1. Wait for the semaphore to be released by the timer callback. 2.Check for messages in the queue (counter of received ++) .3.Sleep for the fixed period.}

8. vSenderTimerCallback function declaration . Function : release semaphore that sender task is waiting for.

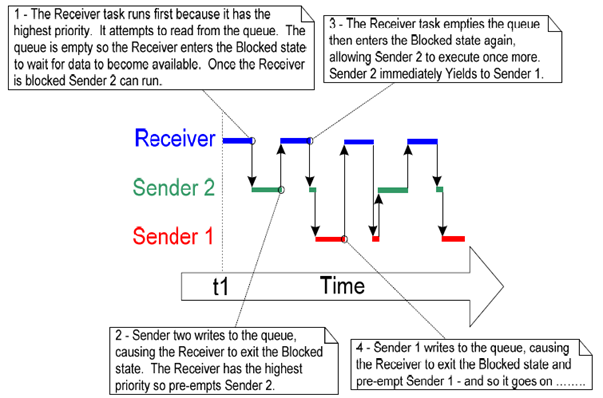
9. vReceiverTimerCallback function declaration. Function: release semaphore then if(receivedMessages >1000) call reset functions .

|  |
| --- |
| void vReceiverTimerCallback(TimerHandle\_t xTimer) { xSemaphoreGive(xReceiverSemaphore);  if (receivedMessages >= 1000) { vResetFunction2(); vResetFunction1();} } |

Design of reset functions :

VResetFunction1 : 1.Reset counters 2.Clear the queue 3.Update the timer periods 4. currentPeriodIndex++; 5.If all values are used (6 iterations = size of array), destroy the timers , print Gameover and stop execution. else Update the timer periods.

VResetFunction2 : 1) print statistics. 2) counter used to calculate average T sender of each iteration =0

 **Fig.1 behaviour of tasks**

## 1.1 Code Snippets: vResetFunction2 including calculation of average T sender of each iteration .

|  |
| --- |
| void vResetFunction2(void) {  // Print the statistics  printf("Total Sent Messages: %d\n", sentMessages[0] + sentMessages[1] + sentMessages[2]);  printf("Total Blocked Messages: %d\n", blockedMessages[0] + blockedMessages[1] + blockedMessages[2]);  printf("Average Time Sender: %lu\n", (totalWaitTime[0] + totalWaitTime[1] + totalWaitTime[2]) / (sentMessages[0] + sentMessages[1] +sentMessages[2]+blockedMessages[0] + blockedMessages[1] + blockedMessages[2]));  for (int i = 0; i < NUMBER\_OF\_SENDERS; i++)  printf("Task %d - Sent: %d, Blocked: %d\n", i, sentMessages[i], blockedMessages[i]);  // make totalWaitTime array ready for the next iteration  totalWaitTime[0]=0; totalWaitTime[1] =0; totalWaitTime[2]=0; |

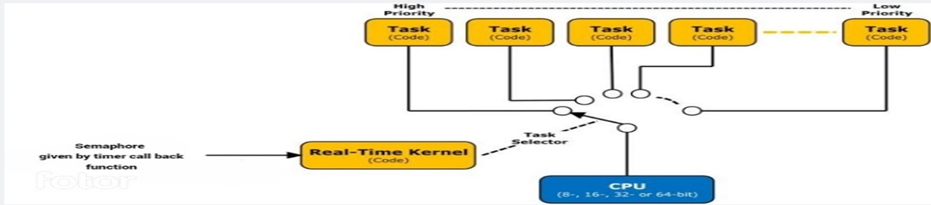
Execution Flow:

The main function initializes all components and starts the FreeRTOS scheduler.

Sender tasks operate in a loop in a random order + receiver task, synchronized by semaphores.

Timers periodically trigger callbacks to release semaphores and manage task execution.

When 1000 messages are received, reset functions are called to handle system resets and print statistics

. Fig.2. usage of semaphores for synchronization.

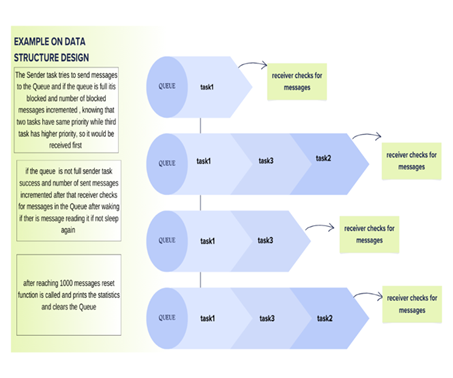
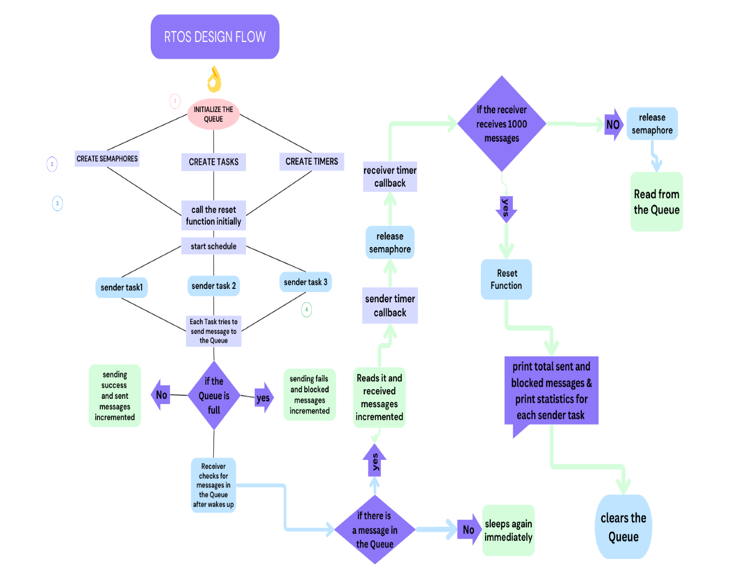


Fig.3 Design flow chart Fig.4 Message Sequence in the queue

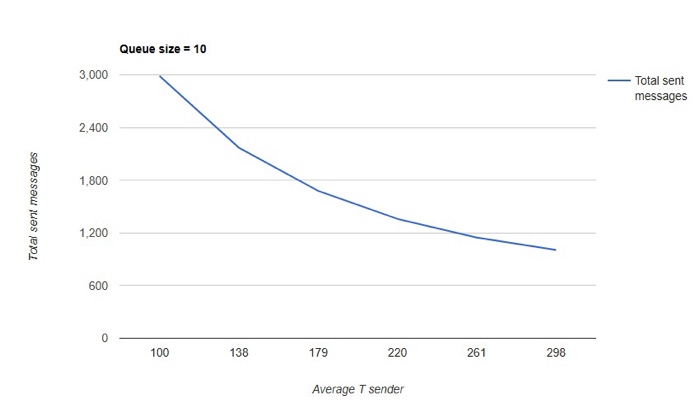
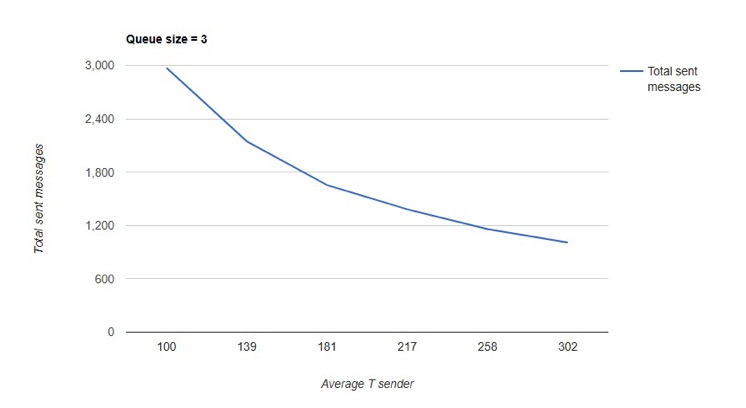
## 2 Results and Discussions: 2.1 Tables and Figures: Table 1:queue size 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Total sent | Total blocked | Avg Tsender | Sent & blocked  by task0 | Sent&blocked  by task1 | Sent &blocked  by task2 |
| 1002 | **1972** | **100** | **Sent:323**  **Blocked:666** | **Sent:351**  **Blocked:635** | **Sent:328**  **Blocked:671** |
| 1002 | **1145** | **139** | **Sent:335**  **Blocked:377** | **Sent:347**  **Blocked:375** | **Sent:320**  **Blocked:393** |
| 1002 | **653** | **181** | **Sent:246**  **Blocked:202** | **Sent:331**  **Blocked:223** | **Sent:325**  **Blocked:228** |
| 1002 | **380** | **217** | **Sent:340**  **Blocked:117** | **Sent:315**  **Blocked:150** | **Sent:348**  **Blocked:113** |
| 1001 | **159** | **258** | **Sent:336**  **Blocked:48** | **Sent:325**  **Blocked:67** | **Sent:340**  **Blocked:44** |
| 1001 | **7** | **302** | **Sent:330**  **Blocked:1** | **Sent:335**  **Blocked:0** | **Sent:336**  **Blocked:6** |

**Table2: queue size 10.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Total sent | Total blocked | Avg T sender | Sent & blocked by task0 | Sent & blocked by task1 | Sent & blocked by task2 |
| 1009 | **1978** | **100** | **Sent:342**  **Blocked: 653** | **Sent:339**  **Blocked:661** | **Sent:328**  **Blocked:664** |
| 1009 | **1162** | **138** | **Sent:342**  **Blocked:383** | **Sent:333**  **Blocked:392** | **Sent:334**  **Blocked:387** |
| 1009 | **661** | **179** | **Sent:373**  **Blocked:219** | **Sent:332**  **Blocked:216** | **Sent:340**  **Blocked:226** |
| 1009 | **351** | **220** | **Sent:347**  **Blocked:112** | **Sent:330**  **Blocked:121** | **Sent:332**  **Blocked:118** |
| 1008 | **138** | **261** | **Sent:341**  **Blocked:44** | **Sent:325**  **Blocked:53** | **Sent:342**  **Blocked:41** |
| 1005 | **1** | **298** | **Sent:355**  **Blocked:0** | **Sent:337**  **Blocked:1** | **Sent:333**  **Blocked:0** |

## 2.2 Graphs and Illustrations: Q1:graph axes : total sent messages vs average T sender

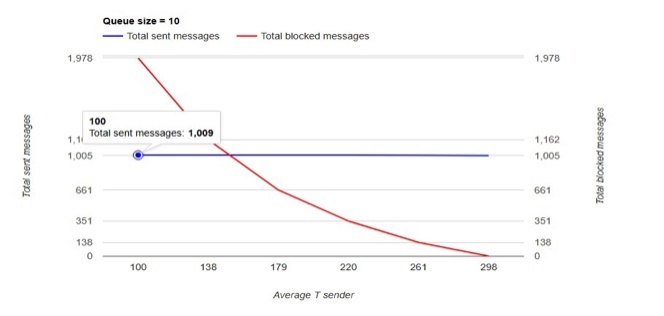
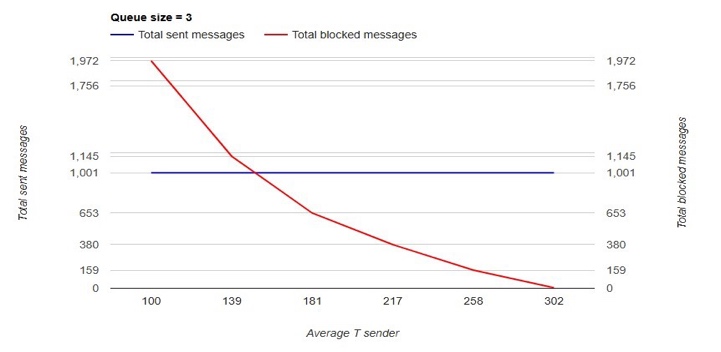


**Queue size =3 Queue size =10**

Comment: total sent messages = successfully sent messages + blocked messages, decaying behaviour because blocked messages decrease gradually as T sender increase as explained before while successfully sent messages seem to be constant

\*The gap between the sent and received messages is obvious at the start and decreases till reaching min. value in the last iteration. \*Explanation: gap =sent messages – received messages = blocked + accumulated messages in queue due to high rate of sending why does the gap decrease? T receiver is constant while T sender is increasing gradually. At first T sender < T receiver (rate of sending is higher) so accumulation of messages is high and blocked messages is higher, in the final iteration, T sender >T receiver (rate of receiving is higher) there is no way for messages to accumulate so blocked messages must decrease because queue will never be full.

**Q2: graph axes :total successfully sent / total blocked messages VS Avg T sender**

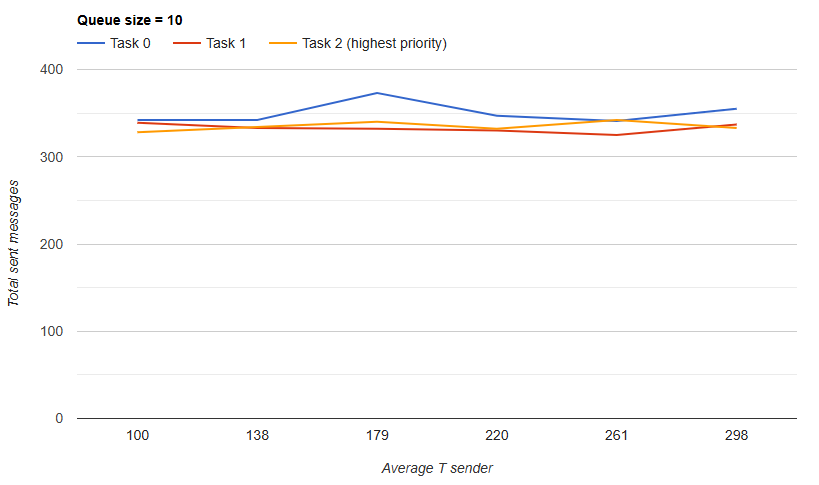
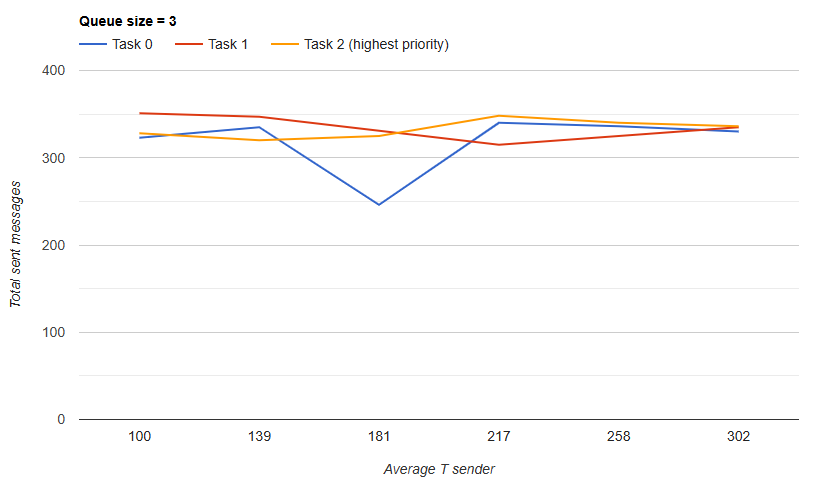
 **Queue size 3 Queue size 10**  


comment: total blocked messages decrease as T sender increase till reaches about 7 blocked messages only.

As T senders increase, now T receiver > T sender, so it is hard to find the queue full, hence blocked messages decrease.

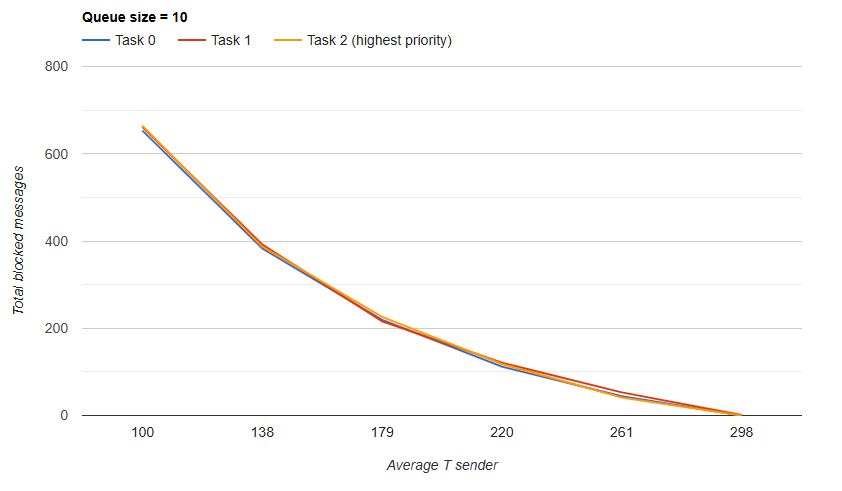
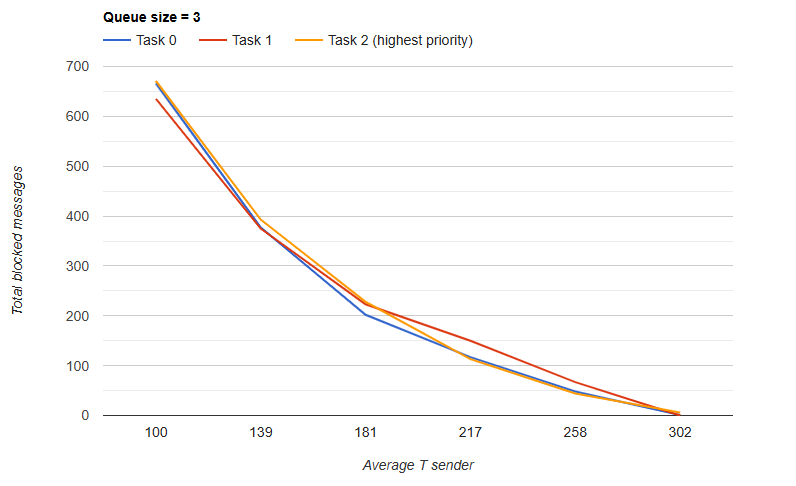
notice that the number of blocked messages decreases by increasing queue size.

**Q3. Graph axes: total sent messages for task 0, task 1, task2 (highest priority) VS average T sender**

**Queue size =3 Queue size =10**

comment: There is no guarantee that blocked messages to be less or sent messages to be higher for higher priority task because sender tasks are sent at random time, hence task priority does not make a great difference. However, if random time was the same by coincidence, the priority is for task2 for sure and task 1 will be blocked. .

**graph axes :total blocked messages for task 0, task 1, task2 (highest priority) VS average T sender**

**Queue size =3 Queue size =10**

comment: they seem to show the same behaviour despite the difference in priority, because they send at random time, so priority does not cause a significant difference. Q-What happens when the queue size increases to 10? Ans: Blocked messages decrease, gap between sent and received messages increase as the size of Queue increases.

**3 References:**

[1] FreeRTOS, *Embedded RTOS Queues*. FreeRTOS. Retrieved June 21, 2024, from <https://www.freertos.org/Embedded-RTOS-Queues.html>

[2] STMicroelectronics. (n.d.). *STM32 Cortex®-M4 MCUs and MPUs Programming Manual*. Retrieved June 21, 2024, from <https://www.st.com/resource/en/programming_manual/pm0214-stm32-cortexm4-mcus-and-mpus-programming-manual-stmicroelectronics.pdf>