1 System Design

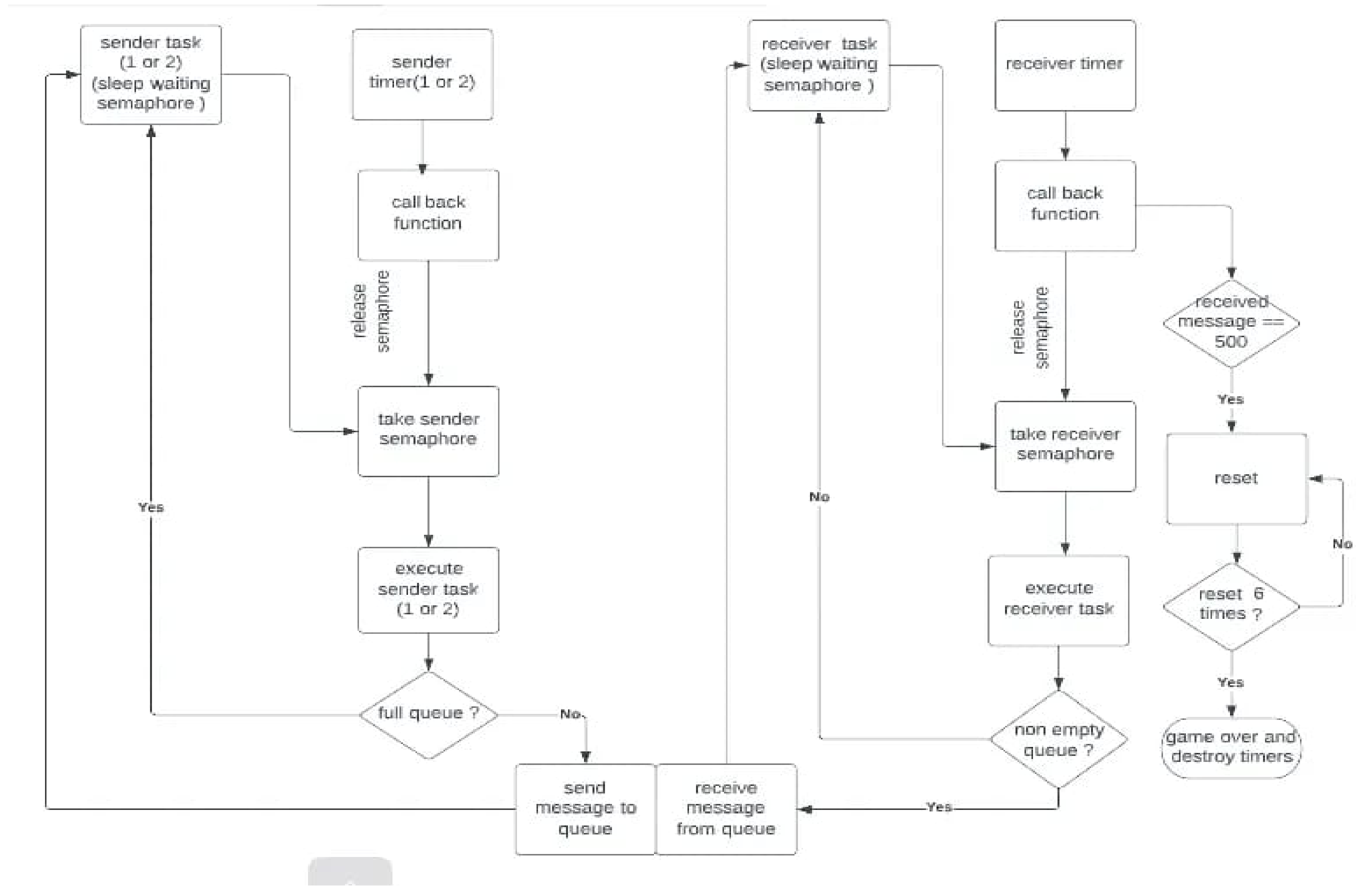


Figure 1: Message Sequence showing object interaction

2 Results and Discussion

Table 1: CQI Index and Corresponding MCS

			Queue Size = 2		**Queue Size = 20**	
TSender1 Q=2, Q=20	**TSender2 Q=2, Q=20**	**Avg TSender**	**Sent Messages**	**Blocked Messages**	**Sent Messages**	**Blocked Messages**
86, 71	**109, 81**	**100**	**502**	**539**	**520**	**795**
87, 114	**178, 190**	**140**	**502**	**354**	**520**	**96**
240, 202	**153, 191**	**180**	**502**	**35**	**520**	**0**
190, 187	**246, 256**	**220**	**502**	**0**	**520**	**0**
179, 337	**326, 198**	**260**	**502**	**0**	**520**	**0**
206, 348	**293, 383**	**300**	**502**	**0**	**520**	**0**

- **The interpretation of the gap between the number of sent and received messages:**
- **<u>when TReceiver > TSender</u>**

- Total sent =502 or 520 to 500 received depend on queue size because the queue stores the messages so when it is full, other sent messages are blocked until the queue has empty space to store again.
- <u>**TReceiver <= TSender**</u>
- The receiver has enough time to receive messages from the queue and blocked messages decrease till reach 0 and the receiving rate be faster than the sending rate so the queue doesn't fill so it is independent on the size of queue.

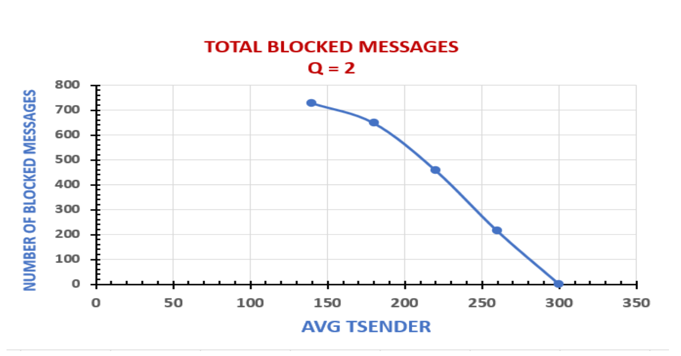
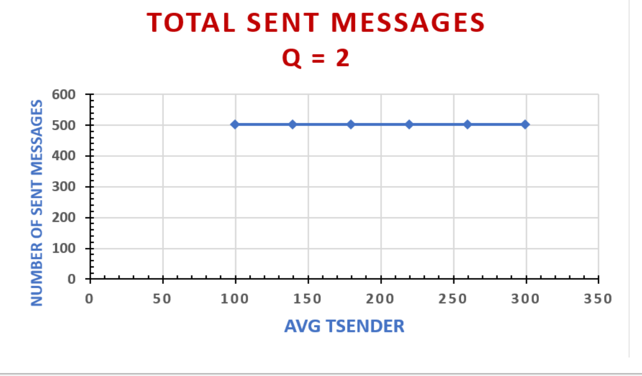
References

Format references as follows

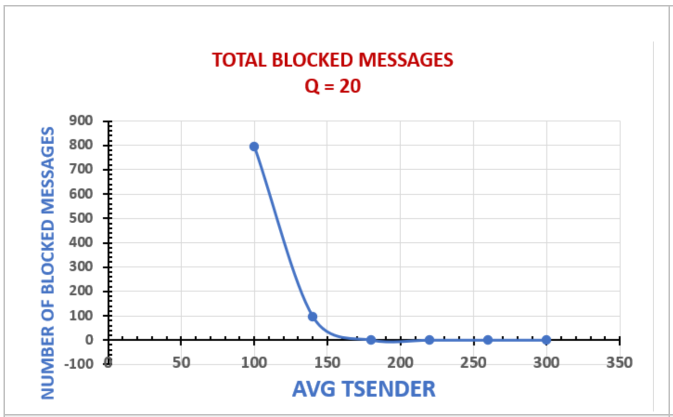
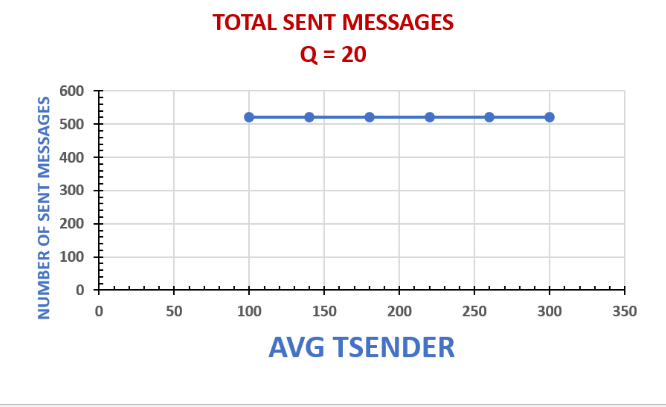
[1] Mastering the FreeRTOS™ Real Time Kernel-Richard Barry.

2.1 Tables and Figures

- Figures
  - Queue Size = 2

	
The Total of Blocked Messages Vs Avg TSender	**The Total of Sent Messages Vs Avg TSender**

  - Queue Size = 20

	
The Total of Blocked Messages Vs Avg TSender	**The Total of Sent Messages Vs Avg Tsender**

- Observation

By increasing Tsender, the blocked messages will decrease till reach 0

Increasing queue size, the blocked message decrease by the difference between the sizes of the two queues (20-2=18) and sent messages increase

2.2 Code Snippets

❖ <u>**Sender timer call back function**</u>

It is periodic timer with period specified by random period function. And after timer fires it calls its callback function which release semaphore for sender task to start execution.

```
static void sender_timer_callback (TimerHandle_t xTimer) {
  xSemaphoreGive(Sender_Sem);}
```

❖ Random period

It is function to get random period for 2 sender tasks from uniform distribution.

```
void random_period() {
for (i = 0; i < 6; i++) {
          Tsender1[i]= (rand() % (upper[i] - lower[i] + 1)) + lower[i];
          Tsender2[i]= (rand() % (upper[i] - lower[i] + 1)) + lower[i];
       }
                                  }
```

❖ Sender tasks

We have 2 sender tasks, each one sleeps for random period then wake up, check if the queue if full
Not full: send message to queue, contain the current time in system ticks
Full: it blocked message
Then sleep for another random period.

```
void sender (void* p)
{
     while(1){
     xSemaphoreTake(Sender_Sem ,portMAX_DELAY);
     char statement [100];

     XYZ = xTaskGetTickCount();
     Sprintf (statement," Time is %d", XYZ);

     if(xQueueSend(globalQueue, &statement, 0)==pdPASS)
        total_send++;
         else
           blocked++;
  }
}
```

❖ Receiver timer call back function

It is periodic timer with period 100ms which release semaphore for receiver task to start receive message, it also checks the number of received message if reached 500 it calls reset function.

```
void Receiver_callback (TimerHandle_t xTimer){
  xSemaphoreGive(Receiver_Sem);
  if( total_rec == 500){
        Reset();
  }
}
```

❖ Receiver task

It receives only one message from queue at a time which contain system ticks from the sender. it sleeps waiting for a semaphore, then wake up and check for non-empty queue:

non empty it will receive message   **empty** sleep for another period of time

```
void receiver (void* p)
{
     while(1){
      if(xSemaphoreTake (Receiver_Sem, portMAX_DELAY)){
    char statement [100];
      if(uxQueueMessagesWaiting(globalQueue) != 0) //checks for messages
          {
           xQueueReceive (globalQueue, &statement, 0); //read the queue
        sprintf ("%s\r\n", statement);
        total_rec++;
           }
        }
    }
}
```

❖ Main

We create 2 sender tasks and one receiver task which have priority higher than sender. then create queue with dictate size and create 3 timers, one timer for each task.

```
int main (int argc, char* argv[])
{      srand(time(NULL));
        globalQueue = xQueueCreate (2, 20* sizeof (char)); //200 - 20
    Receiver_Sem = xSemaphoreCreateBinary ();
    Sender_Sem1 = xSemaphoreCreateBinary ();
    Sender_Sem2 = xSemaphoreCreateBinary ( );
    If (globalQueue! = NULL) {
    xTaskCreate (sender,"TSender1" ,1024, (void*) 0, 0, &sender1Handle);
    xTaskCreate (sender,"TSender2" ,1024, (void*) 0, 0, &sender2Handle);
    xTaskCreate (receiver, "TReceiver" ,1024, (void*) 0, 1, &receiverHandle);
    }
    timer_sender1   =   xTimerCreate   ("Timer_sender1",   (pdMS_TO_TICKS   (1000)),   pdTRUE,   (void   *)   0,
sender_timer_callback);
    timer_sender2   =   xTimerCreate   ("Timer_sender2",   (pdMS_TO_TICKS   (1000)),   pdTRUE,   (void   *)   2,
Receiver_callback);
    timer_receiver   =   xTimerCreate   ("Timer_receiver",   (pdMS_TO_TICKS   (3000)),   pdTRUE,   (void   *)   1,
sender_timer_callback);
    Reset();
   If ((timer_sender1! = NULL) && (timer_sender2! = NULL) && (timer_receiver! = NULL))
   {
     timer_sender1Started = xTimerStart (timer_sender1, 0);
     timer_sender2Started = xTimerStart (timer_sender2, 0);
     timer_receiverStarted = xTimerStart (timer_receiver, 0);
   }
   If (timer_sender1Started == pdPASS && timer_sender2Started == pdPASS && timer_receiverStarted == pdPASS)
   {
     vTaskStartScheduler();
   }

   return 0;}
```

❖ Reset

It prints total number of successful and blocked message then reset the counter of them, change sender time, if reset 6 times it prints game over and will end scheduler and destroy timers.

```
void Reset()
{        if (total_rec == 500) {
      printf ("Total number of sent messages = %d \r\n", total_send);
    printf ("Total number of Blocked messages = %d \r\n", blocked);
    blocked =0; // Reset all the counters
    total_send =0;
    total_rec =0;
    xQueueReset (globalQueue); //Clear Queue
      }
    if (j >5)// If we used all values in the array, destroy the timers and print a
message “Game Over” and stop execution
      {
        XTimerDelete (timer_sender1, 0); // Destroy the Timers
        xTimerDelete (timer_sender2, 0);
        xTimerDelete (timer_receiver, 0);
        printf ("Game Over\r\n");
        vTaskEndScheduler();
        return;
      }
    random_period();
        xTimerChangePeriod (timer_sender1, Tsender1[j], (TickType_t) 0);
        xTimerChangePeriod (timer_sender2, Tsender2[j], (TickType_t) 0);
        printf("Tsender1 is %d\r\n",Tsender1[j]);
        printf("Tsender2 is %d\r\n",Tsender2[j]);
        j++;
}
```