1 System Design

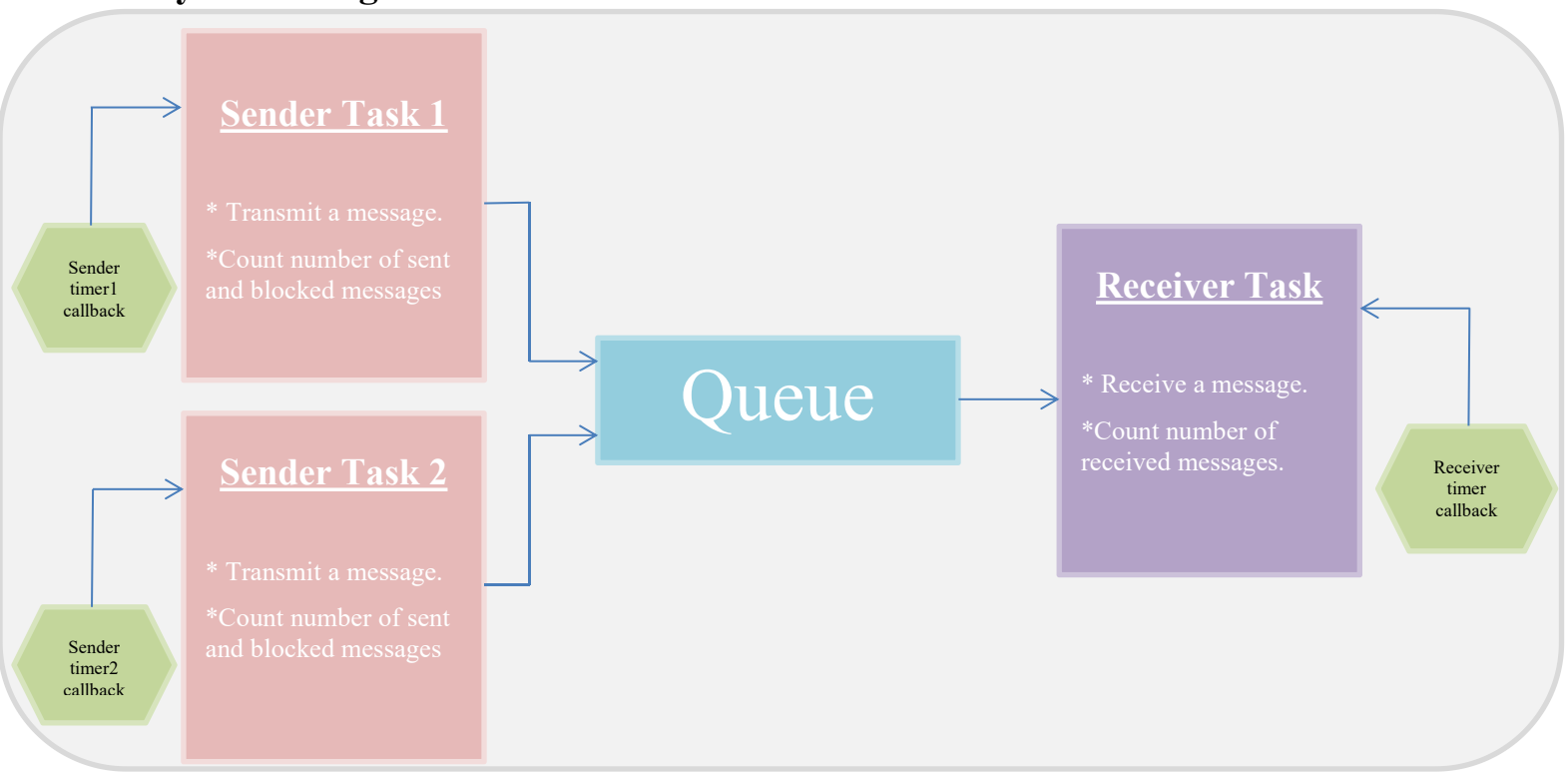

Figure 1: System Design.

Mainly, the system consists of four different structures :

- A message queue: which represents the inter-task communication
- Two sender tasks : each of them is dedicated to send a message to the queue every Tsender ms. Tsender changes after every sending operation and is drawn randomly from a uniform-distribution function. The upper and lower bounds for this uniform distribution are taken from two arrays, one for each.
- One receiver task : this task is dedicated to check if there is any data in the queue to receive every Treceiver=100 ms.
- Three timers: one for each task, controlling the sending and receiving operations by giving the task, that the timer is created for, the permission to do their mission or in other words, releasing a dedicated semaphore. This permission is given through calling their own callback function each time they fire.

More detailed information and code explanation:

- Uniform_distribution function

```
int uniform_distribution(int LowerBound, int UpperBound) {
    int range = UpperBound - LowerBound + 1;
    double random_number = rand()/(1.0 + RAND_MAX);
    int value = (random_number * range) + LowerBound;
    return value;
}
```

It takes the upper and the lower bounds for our desired range, generates a random number using rand(), divides this number by the max number this rand function can send added to one so that we always get a number that is less than one, we multiply this new number by our range so that we get a portion of it that represents how fare we are fron the lower bound and add this new value to our lower bound.
This algorithm is used to draw a different Tsender every sending operation.

- Handle timer period functions

```
void handle_timer_1_period(void){
   // draw a random period from the uniform distribution
   Tsender=uniform_distribution(lower,upper);
   xTimerChangePeriod(sender_1_Timer, pdMS_TO_TICKS(Tsender) , 0);
```

```
}
```

This function updates Tsender with a random value based on the latest values assigned for the upper and lower limits, then change the timer period with this new value. Also, there is another one handling the second timer that controls the other sender task.
This function is called at the end of the callback function dedicated for each timer.

- Sender tasks

```
void sender_1_Task(void *p)
{
 char Message[BUFFER_SIZE];
 sender_1_Semaphore = xSemaphoreCreateBinary();
 BaseType_t status;

 while(1)
 {
   xSemaphoreTake( sender_1_Semaphore, portMAX_DELAY );
   sprintf(Message, "Time is %d", xTaskGetTickCount() );
   status = xQueueSend(myQueue, Message, 0);
   if ( status == pdPASS ){
     sent_messages++;
   }

   else{
     blocked_messages++;
   }
 }
}
```

Each sender tasks creates a dedicated semaphore, then inside an infinite loop, the sender tasks tries to take the semaphore but it cannot take it unless the timer call back function releases it so it sleeps. When the semaphore is released after Tsender, it sends a message containing time in ticks to the queue assuming there is space in the queue, if not , then message is blocked.

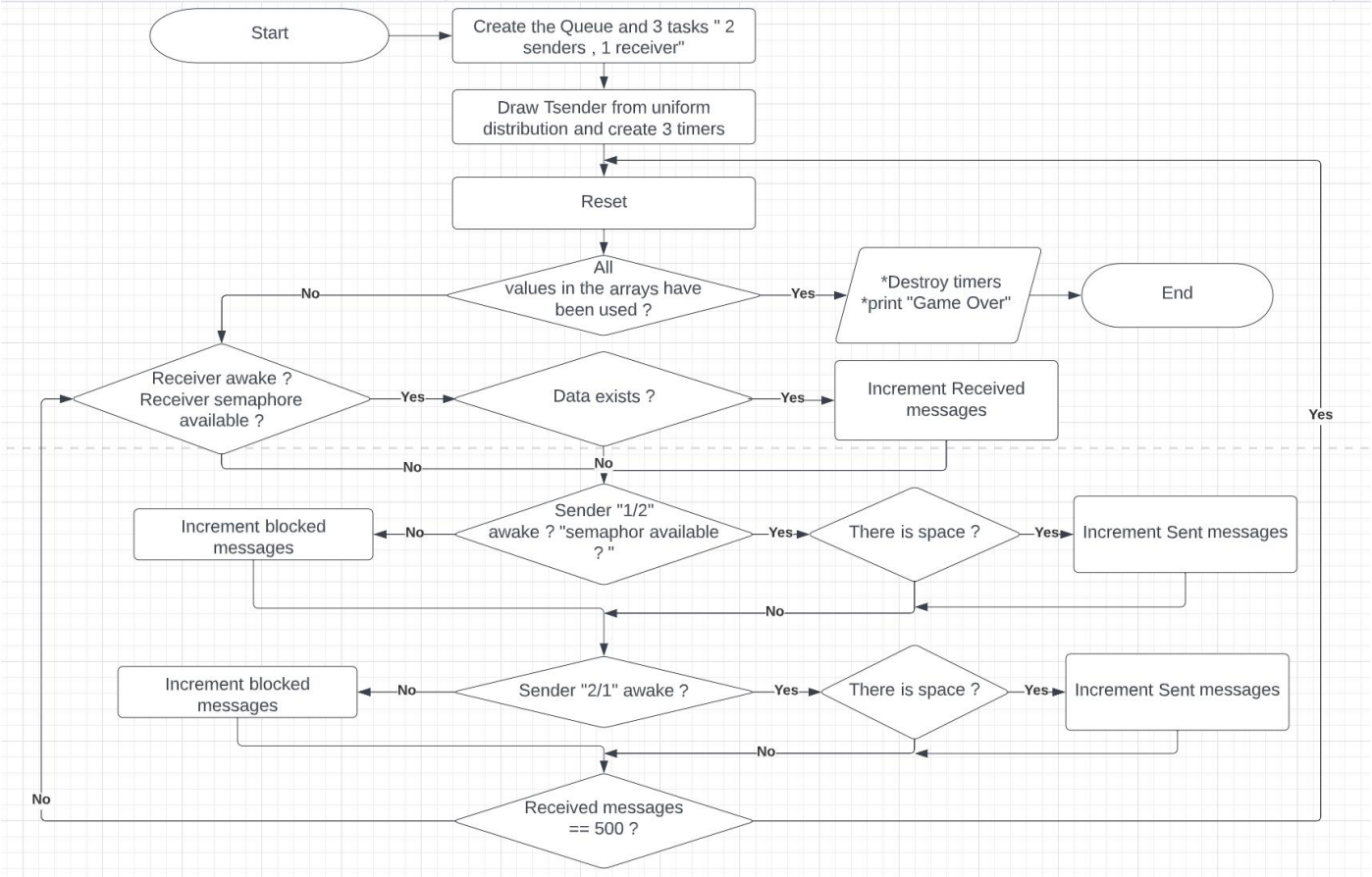

Figure 2: flowchart of the whole system

2 Results and Discussion

```
Other event 0x302
number of successfully sent Messages = 502
number of Blocked Messages = 490

number of successfully sent Messages = 502
number of Blocked Messages = 211

number of successfully sent Messages = 502
number of Blocked Messages = 64

number of successfully sent Messages = 501
number of Blocked Messages = 0

number of successfully sent Messages = 501
number of Blocked Messages = 0

number of successfully sent Messages = 500
number of Blocked Messages = 0

Game Over!


QEMU semihosting exit(0)
```

Figure 3: output when queue size is

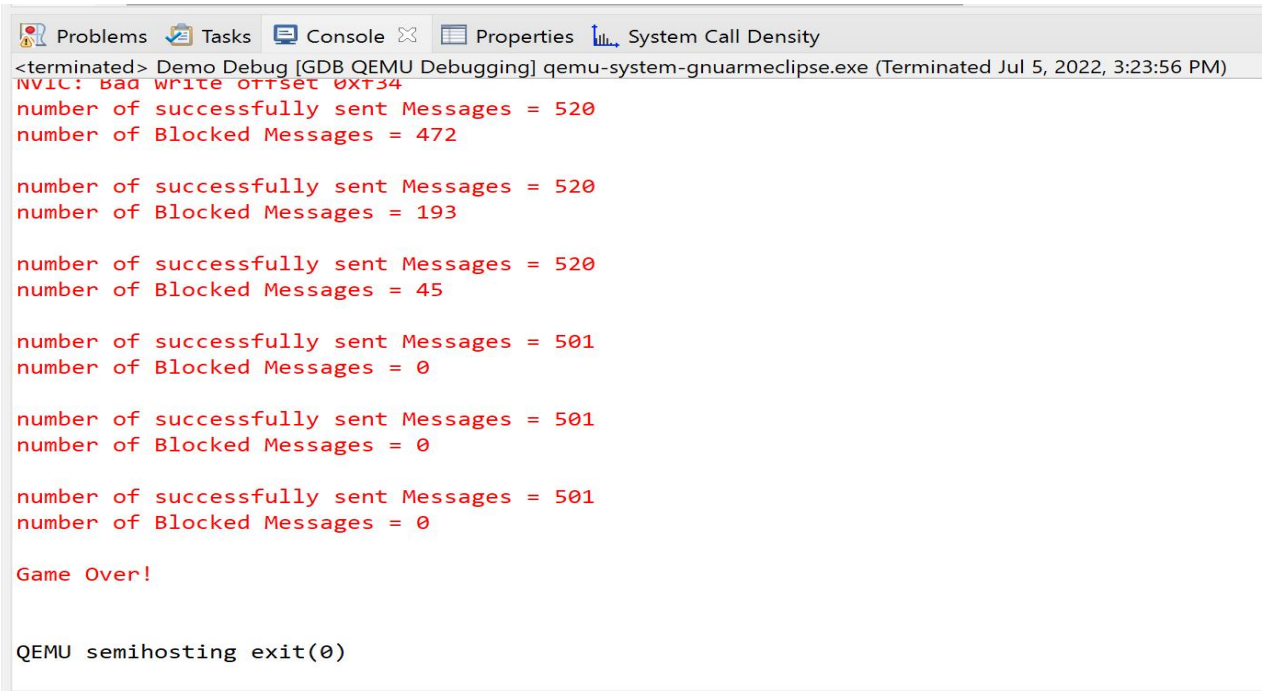

Figure 4: output when queue size is 20

Figure 5: the number of blocked and received messages as a function of average value of Tsender for queue size=2

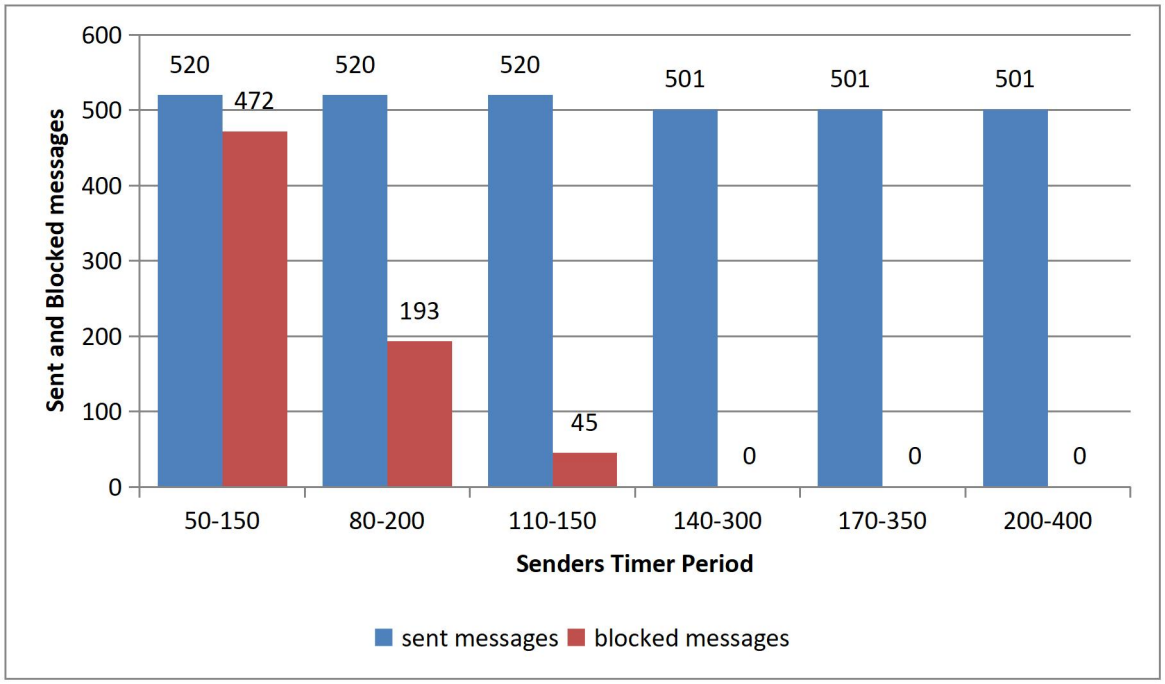

Figure 6: the number of blocked and received messages as a function of average value of Tsender for queue size=20

Discussion:

- What happens when queue size increases?

It is pretty obvious that when the size of the queue increased to 20, the number of blocked messages decreased, and this is because larger size means larger space in the queue which decreases the rate by which the probability of a message getting blocked when the sender task is awake since the time by which the queue is full, when messages start to get blocked , is stretched.

As we can see from Figure 3 and Figure 4 : for the first batch, blocked_messages =490 when size =5 and decreased to 472 when size is increased to 20.

- The gap between number of sent and received messages explanation:

The reason why we have a gap between the sent and received messages, which means that the number of sent messages and the number of received ones are not equal or that not all the message that we send we can receive, is the different rate the sender and receiver tasks operate at:

- If Tsender is guaranteed to be less than Treceiver doubled, since we have two sender tasks, the queue gets messages more frequently than it gets red of them, so once the queue is full, some messages get blocked and less number of messages than before is successfully sent to the queue. Finally, we end up with total number of sent messages greater than max received messages by queue size. Where max received messages =500 in our case.

  -As we can see from Figure 3 and Figure: for the first three batches, sent_messages= max received messages+ queue size.

- If Tsender is guaranteed to be larger than Treceiver doubled, the queue checks for messages to receive more frequently than it gets new messages, so when Treceiver is not small enough we might end up with one excess message that we don't receive.

  -As we can see from Figure 3 and Figure: for the fifth batch for example, sent_messages= max received messages+ 1.

References

– Richard Barry. "Interrupt Management" in *Mastering the FreeRTOS Real Time Kernel - a Hands On Tutorial Guide*. Real Time Engineers Ltd, 2016, pp.217-228.