1 System Design

Our system design consists of some main components:

1. 3 Sender tasks, one of them has higher priority than others, and a Receiver task, which has the highest priority.
2. A queue to handle communication between all tasks.
3. Timer for each task, each timer has its callback function, which manages the sleep or wake state of the tasks.
4. Semaphore for each task, which blocks or unblocks the tasks from doing its task.

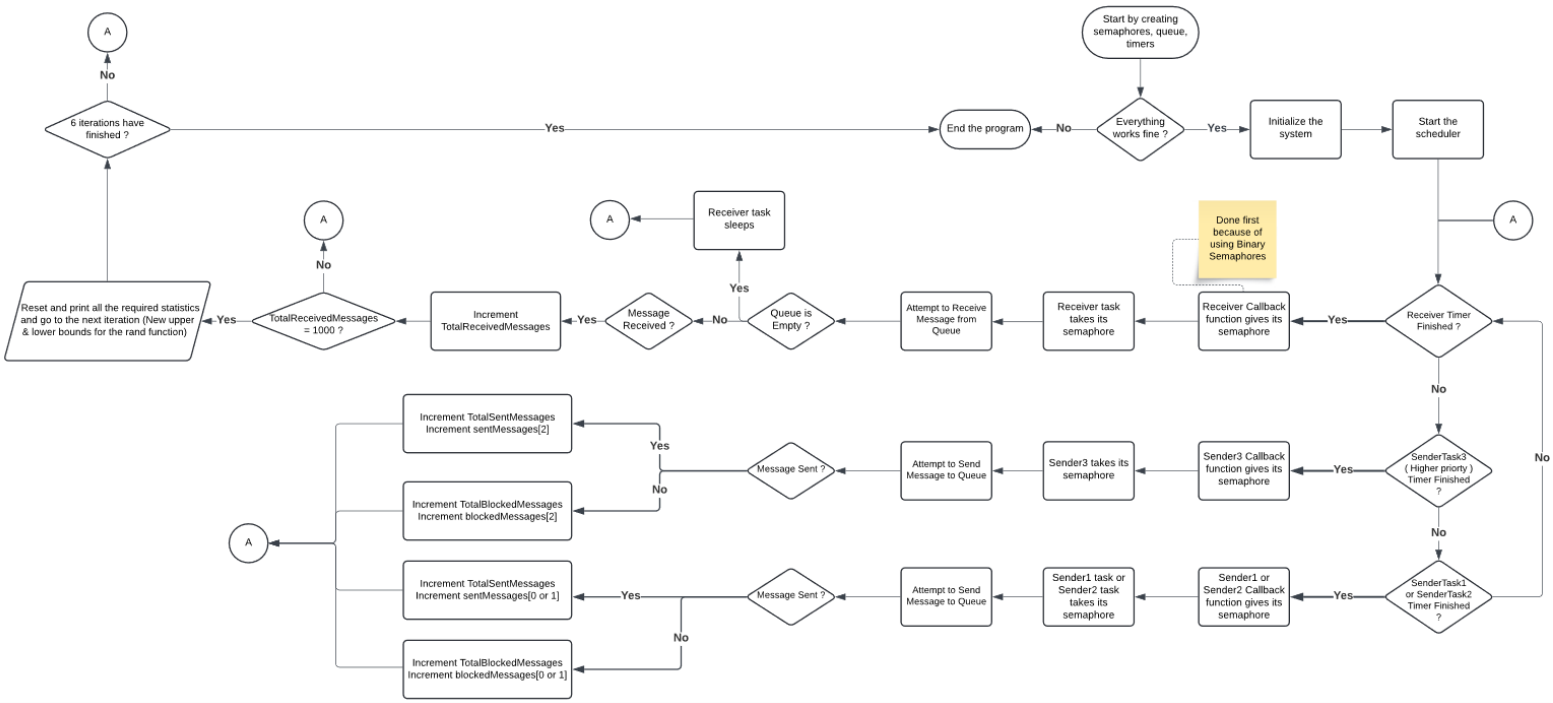

Figure 1: Our system flow

We start by first creating our main components and initialize the system, then starting the scheduler, and now we wait to see which task timer , will finish first and in each one there will be a callback function that will give the task's semaphore, then the task takes the semaphore and either try to send to the queue (if a sender task) or receive (if a receiver task) from the queue and we repeat this process until the total received messages is 1000 and then iterate again for another 5 times, that will be 6 iterations. In each iteration, then we change the lower and upper bounds of the uniform distribution random function to a higher value to see the effect of changing the timer period of the sender tasks, while the receiver task is fixed at 100ms.

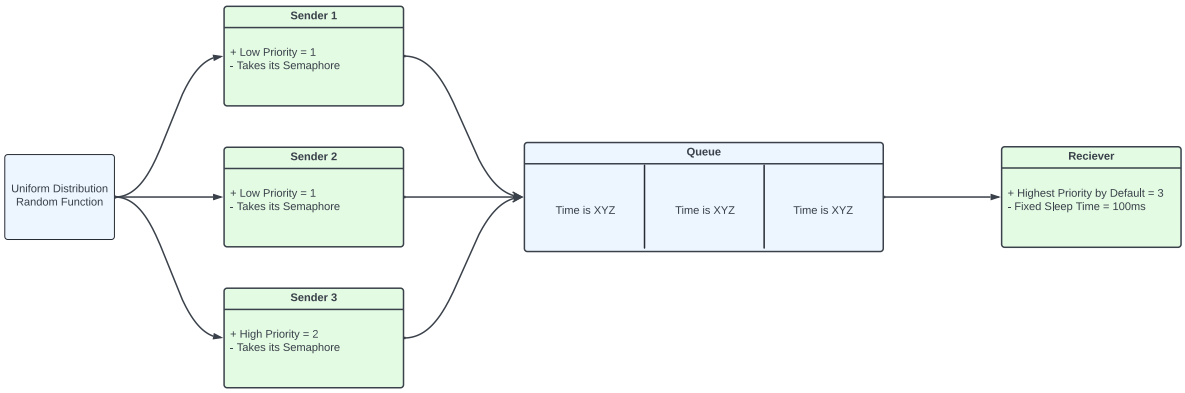

Figure 2: Handling Queue

We created three sender tasks in one function using parameter sent in xTaskCreate() function, instead of making three functions. And we chose the parameters to be 0,1,2, which was extremely useful in dealing with arrays, as now the parameters of each task are the index of its arrays.

```
void SenderTask(void *Parameters)
{
    int taskId = (int) Parameters;
    BaseType_t Send_Status;
    char message[20];
    while (1)
    {
        xSemaphoreTake(senderSemaphore[taskId], portMAX_DELAY);
        TickType_t period = rand() % (UpperBound[timerIndex] - LowerBound[timerIndex] + 1) +
                            LowerBound[timerIndex];
        xTimerChangePeriod(senderTimers[taskId], pdMS_TO_TICKS(period), 0);
        total_time[taskId] = total_time[taskId] + period;

        snprintf(message, sizeof(message), "Time is %lu", xTaskGetTickCount());
        Send_Status = xQueueSend(messageQueue, &message, 0/*Wait Time*/);
        if (Send_Status == pdPASS)
        {
            sentMessages[taskId]++;
            totalSentMessages++;
            printf("%.20s.\n",message);
        }
        else
        {
            printf("Couldn't send to the queue.\n");
            blockedMessages[taskId]++;
            totalBlockedMessages++;
        }
    }
}
```

And same also for the sender tasks callback function, we used one function instead of using three functions. Using the parameter trick.

```
void SenderTimerCallback(TimerHandle_t xTimer)
{
    int taskId = (int) pvTimerGetTimerID(xTimer);
    //printf("Callback number %d executed\n",taskId);  //Used for debugging
    xSemaphoreGive(senderSemaphore[taskId]);
}
```

2 Results and Discussion

Iterations	Sent Messages			Blocked Messages			Average Sender Time (ms)		
	Sender 1	Sender 2	Sender 3	Sender 1	Sender 2	Sender 3	Sender 1	Sender 2	Sender 3
1	**332**	**332**	**338**	**679**	**669**	**684**	**98**	**99**	**97**
2	**338**	**322**	**342**	**381**	**397**	**369**	**139**	**139**	**140**
3	**318**	**336**	**348**	**234**	**216**	**204**	**181**	**181**	**181**
4	**331**	**337**	**333**	**123**	**111**	**118**	**220**	**222**	**221**
5	**341**	**338**	**323**	**45**	**45**	**62**	**259**	**261**	**259**
6	**329**	**334**	**338**	**6**	**5**	**2**	**304**	**300**	**299**

Table 1: Queue with Size 3

Observation: As the numbers show, Increasing the Average Sender Time results is less blocked messages.

Iterations	Sent Messages			Blocked Messages			Average Sender Time (ms)		
	Sender 1	Sender 2	Sender 3	Sender 1	Sender 2	Sender 3	Sender 1	Sender 2	Sender 3
1	**328**	**327**	**354**	**672**	**693**	**660**	**99**	**98**	**98**
2	**340**	**320**	**349**	**376**	**400**	**364**	**139**	**138**	**140**
3	**323**	**336**	**350**	**233**	**216**	**198**	**180**	**181**	**182**
4	**341**	**328**	**339**	**106**	**123**	**116**	**223**	**221**	**219**
5	**344**	**329**	**336**	**38**	**57**	**50**	**261**	**259**	**259**
6	**331**	**335**	**335**	**0**	**0**	**0**	**304**	**300**	**301**

Table 2: Queue with Size 10

Observation: When the Queue Size is increased the Number of Blocked Message is less in every iteration & at the last iteration The number of Blocked Messages is Zero as the Receiver Time (100ms), and the average sender time is above (300ms) so the queue now receives faster than the senders send which result in no full queue at any time.

Iterations	Queue Size = 3		Queue Size = 10	
	Total Sent Messages	Total Blocked Messages	Total Sent Messages	Total Blocked Messages
1	**1002**	**2032**	**1009**	**2025**
2	**1002**	**1147**	**1009**	**1140**
3	**1002**	**654**	**1009**	**647**
4	**1001**	**352**	**1008**	**345**
5	**1002**	**152**	**1009**	**145**
6	**1001**	**13**	**1001**	**0**

Table 3: Total Number of Messages

Observation: The gap in each iteration between the total received messages (1000) and the total sent messages is because of maybe when the message number 1000 sent to the queue, the queue does not receive it immediately, as they all have timers with different periods, so senders may send like 1 or 2 messages until the queue receive the message number 1000.

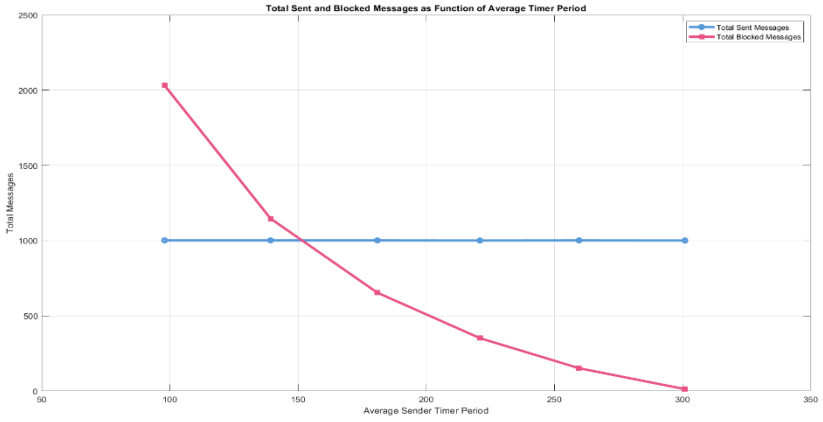

Figure 3: Total Number of Messages Function of Average Timer Period

2.1 Queue with Size 3:

2.1.1 High Priority Sender:

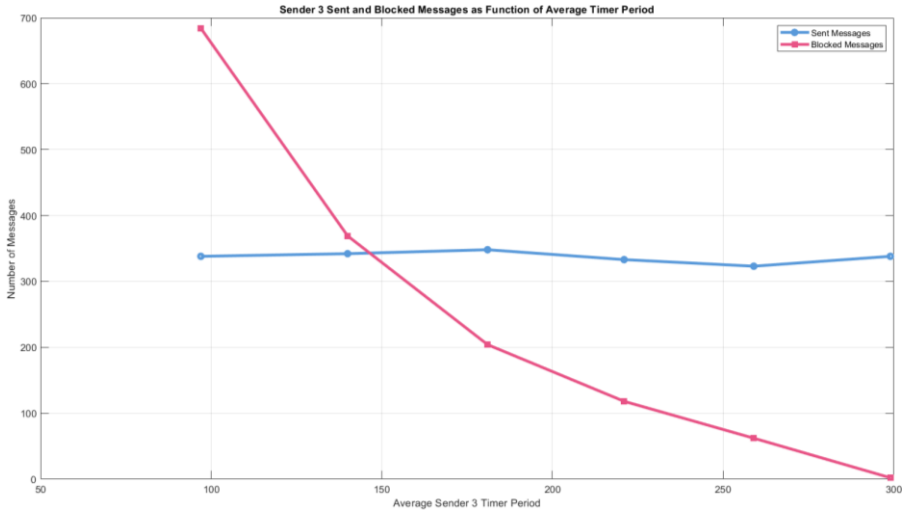

Figure 4: Sender 3 (High Priority) Number of Messages Function of its Average Timer Period

2.1.2 Low Priority Sender:

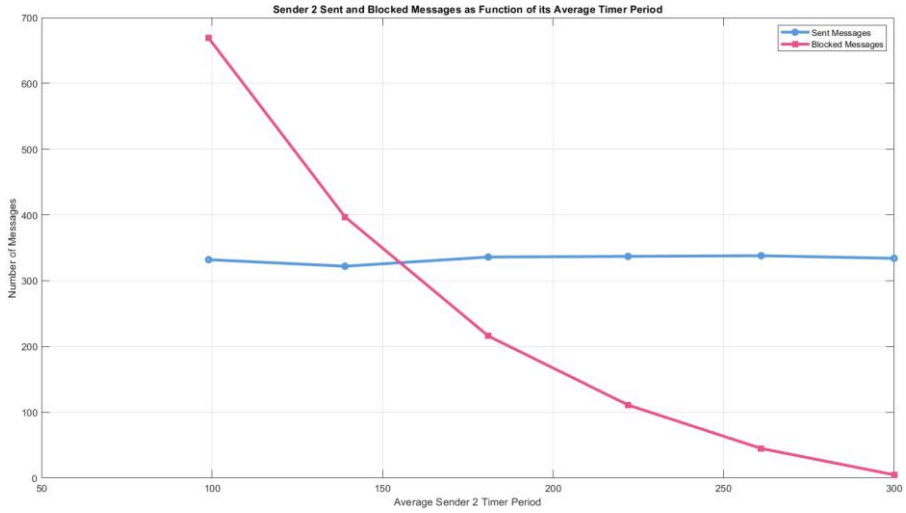

Figure 5: Sender 2 (Low Priority) Number of Messages Function of its Average Timer Period

2.2 Queue with Size 10:

2.2.1 High Priority Sender:

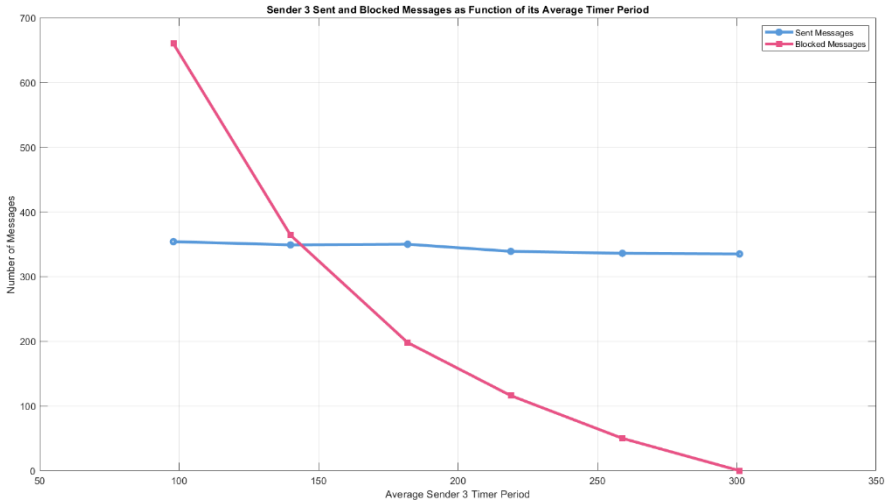

Figure 6: Sender 3 (High Priority) Number of Messages Function of its Average Timer Period

2.2.2 Low Priority Sender:

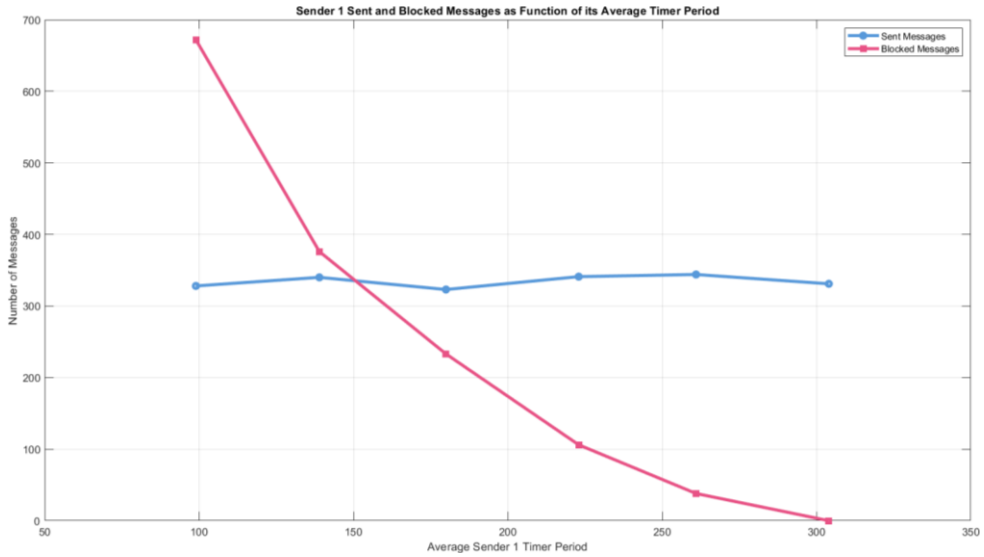

Figure 7: Sender 1 (Low Priority) Number of Messages Function of its Average Timer Period

References

[1] R. Barry, "FreeRTOS User Manual," Version 10.0.0, Real Time Engineers Ltd. [Online]. Available: **https://www.freertos.org/Documentation/FreeRTOS-documentation-and-book.html**

[2] "FreeRTOS task communication and synchronisation with queues, binary semaphores, mutexes, counting semaphores and recursive semaphores," . [Online]. Available: **https://www.freertos.org/Inter-Task-Communication.html**