|  |  |  |  |
| --- | --- | --- | --- |
| BN | Section | Student ID | اسم الطالب |
| **15** | **3** | **9220637** | **مايكل هاني سمير جرجس** |
| **19** | **4** | **9220889** | **مينا ايهاب منصور صليب** |

# System Design

### System flow

The following figure shows the flow and the logic of our system:

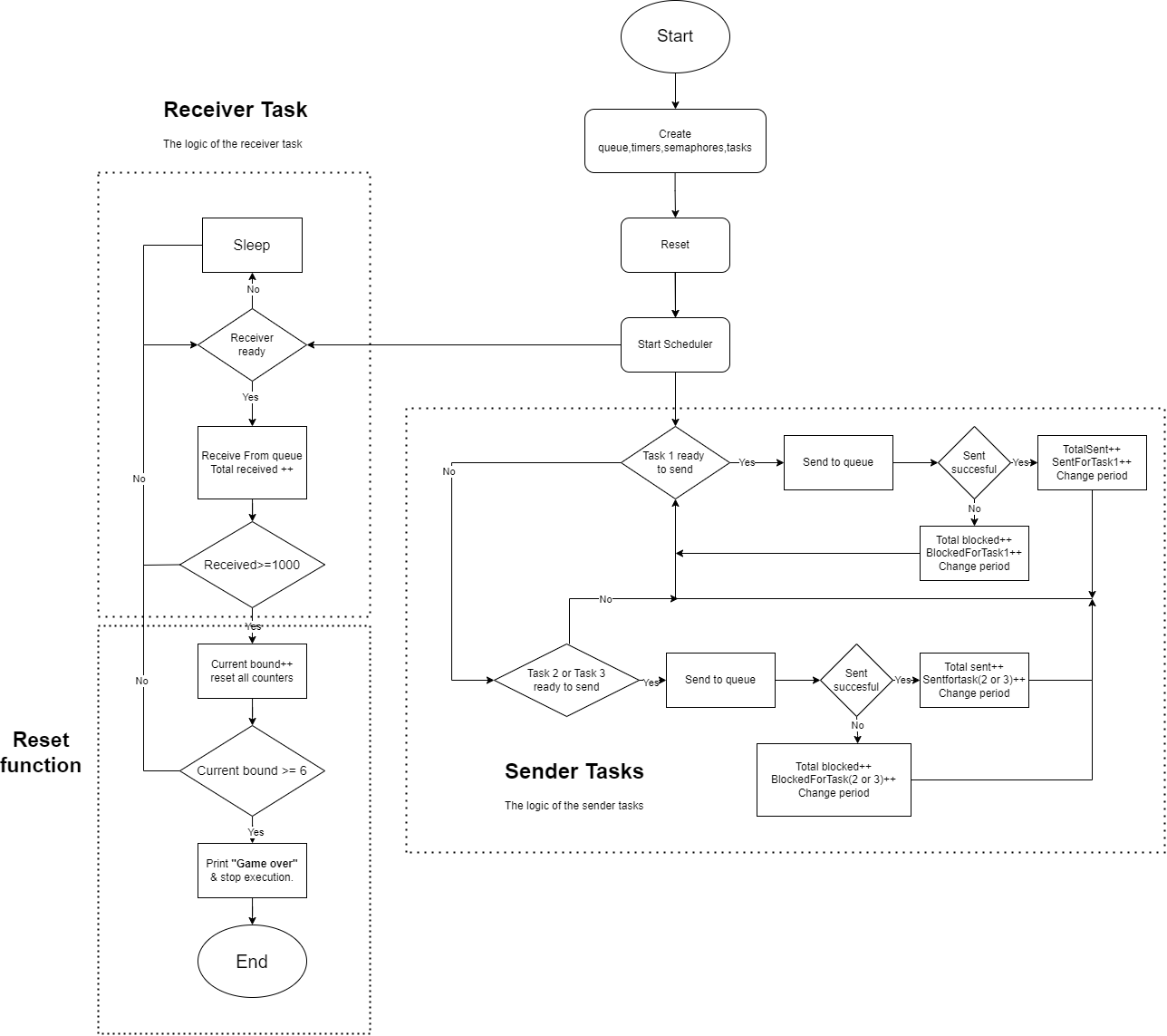


Figure 1: System flow

### 1.2 Queue visualization

The figure shows the interactions between the tasks and the queue and the priority of each task.

# 

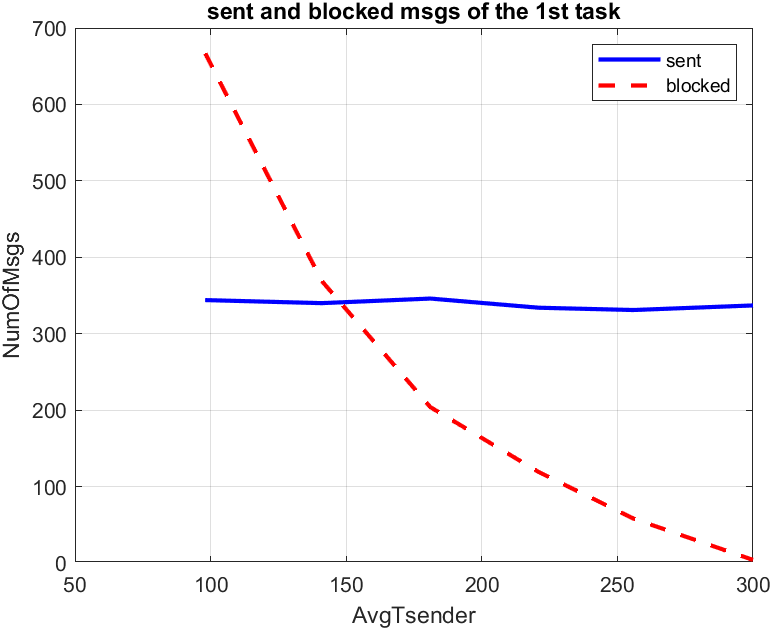
Figure 2: Handling the queue.

# Results and Discussion

### 2.1 Queue of size 3statistics and Conclusions.

| Iterations | Sender 1 (high priority) | | | Sender 2 | | | Sender 3 | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sent | Blocked | Average Time | Sent | Blocked | Average Time | Sent | Blocked | Average Time |
| 1 | 344 | 667 | 98ms | 327 | 686 | 98ms | 331 | 673 | 99ms |
| 2 | 340 | 369 | 141ms | 345 | 378 | 138ms | 317 | 404 | 138ms |
| 3 | 346 | 204 | 181ms | 327 | 223 | 181ms | 329 | 224 | 180ms |
| 4 | 334 | 119 | 221ms | 346 | 115 | 216ms | 322 | 131 | 220ms |
| 5 | 331 | 58 | 256ms | 332 | 56 | 258ms | 339 | 55 | 253ms |
| 6 | 337 | 4 | 300ms | 335 | 6 | 299ms | 330 | 4 | 305ms |

Table 1: Statistics for Queue of size 3

A graph with a line and a red line

Description automatically generated

Figure 4: Lower priority Messages

Figure 3: Higher priority Messages

The above graphs and table show the sent and blocked messages in each task independently and our conclusions from them are:

* The blocked always starts at high numbers because the queue is being filled at a faster rate compared to the rate of the receiver receiving the messages.
* The gap between the received and the sent keeps going down as the AvgTsender increases because it now has enough time to receive.

### 2.2 Comparison between queue size 3 and 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iterations | All Senders | | | |
| Total sent | | Total blocked | |
|  | Queue with Size 3 | Queue with Size 10 | Queue with Size 3 | Queue with Size 10 |
| 1 | 1002 | 1009 | 2026 | 1976 |
| 2 | 1002 | 1009 | 1151 | 1128 |
| 3 | 1002 | 1009 | 651 | 649 |
| 4 | 1002 | 1009 | 365 | 352 |
| 5 | 1002 | 1009 | 169 | 137 |
| 6 | 1002 | 1002 | 14 | 0 |

Table 2: Statistics for queue of size 3 and 10

A graph of a line graph

Description automatically generatedA graph with a line and a red line

Description automatically generated

Figure 5: Total messages when queue size is 3

Figure 6: Total messages when queue size is 10

The above graphs and table show the sent and blocked messages when using different sized queues and our conclusions from them are:

* The blocked messages when using sized 10 queue is less than compared to sized 3 queue because it now has more space to store more data so the blocking decreases.

# 3.0 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>

# 4.0 Code Snippets

|  |
| --- |
| void SenderTask(void \*Parameters){  int senderID = (int) Parameters;  char message[MsgSize];  BaseType\_t xStatus;  RandomPeriodSum[senderID]=StartPeriod[senderID];  RandomPeriodCount[senderID]++;  while (1) {  xSemaphoreTake(senderSemaphores[senderID], portMAX\_DELAY);  TickType\_t CurrentTime = xTaskGetTickCount();  snprintf(message, MsgSize, "Time is %lu", CurrentTime);  xStatus=xQueueSend(MessageQueue, &message, 0);  if ( xStatus == pdPASS) {  sentMessages[senderID]++;  totaltransmittedmessages++;  } else {  blockedMessages[senderID]++;  totalBlockedMessages++;  }  int32\_t NewPeriod = (rand() % (Max\_Bounds[CurrentRangeIndex] - Min\_Bounds[CurrentRangeIndex] + 1)) + Min\_Bounds[CurrentRangeIndex];  xTimerChangePeriod(senderTimers[senderID], pdMS\_TO\_TICKS(NewPeriod), 0);  RandomPeriodSum[senderID] += NewPeriod;  RandomPeriodCount[senderID]++;  }} |

The code above shows what the three tasks do, tasks wait for the timer to be fired and the callback function to be executed to take the semaphore and sends a message containing the current time to the queue, after finishing the counters are incremented and a new random period will be set for each task.

|  |
| --- |
| void reset(void) {  for (int i = 0; i < 3; i++) {  AvgOneTask[i] = RandomPeriodSum[i] / RandomPeriodCount[i];}  printf("Total sent messages are %d\n", totaltransmittedmessages);  printf("Total blocked messages are %d\n", totalBlockedMessages);  for (int i = 0; i < 3; i++) {  int x = i+1;  printf("Average time of Task %d is %d ms\n",x, AvgOneTask[i]);}  TotalAvg=(AvgOneTask[0]+AvgOneTask[1]+AvgOneTask[2])/3;  printf("Average time of three tasks is %d ms\n",TotalAvg);  for (int i = 0; i < 3; i++)  {  int x = i+1;  printf("for task %d The transmitted messages: %d, Blocked ones: %d\n", x, sentMessages[i], blockedMessages[i]);  sentMessages[i] = 0;  blockedMessages[i] = 0;  }  totaltransmittedmessages = 0;  totalBlockedMessages = 0;  receivedMessages = 0;  for (int i = 0; i < 3; i++) {  RandomPeriodSum[i] =0;  RandomPeriodCount[i]=0;}  xQueueReset(MessageQueue);  CurrentRangeIndex++;  if ( CurrentRangeIndex>= 6) {  for (int i = 0; i < 3; i++) {  xTimerDelete(senderTimers[i], 0);  }  xTimerDelete(receiverTimer, 0);  printf("Game Over\n");  exit(0);  vTaskEndScheduler();  }  } |

The code above shows what the Reset function do, it does the following in order:

* Calculates the average period for each task
* Prints Total Messages (sent & blocked)
* **Prints Average Time for Each Task**.
* **Calculates and Print Overall Average Time.**
* **Prints and resets all Counters**.
* Resets the Queue
* Increment the range index
* **Checks for Termination Condition**: