| اسم الطالب | رقم الجلوس | الفصل | رقم المقعد | رقم المجموعة على البلاكبورد |
|---|---|---|---|---|
| **احمد أسامة عبد المقصود على** | **32001** | **1** | **1** | **97** |
| **حسام على احمد محمد** | **32056** | **1** | **56** | **97** |

# 1) System Design

## Global variables:

The counters are declared and set to zero.

An array containing the periods of the sender named Tsender is declared with its index set to negative 1.

Handlers are declared for the Queue and all the Timers, Semaphores and Tasks created.

## Main() function:

The two timers, SenderTimer and ReceivedTimer, are declared. Both timers are periodic having periods an array of [100,140,180,220,260,300] and 200 msec respectively.

Two tasks are then created, Sender and receiver, both with allocated memory of 200 and having the same priority.

Then two semaphores are created for the Sender and receiver tasks. The two semaphores work with a tick time of 10000 and are taken once created in the main function.

If both timers are created successfully, the timers start immediately which will be followed by the init function.

## Init() function:

The number of sent and blocked messages are printed.

The global variables' values will be reset and the index of the array will be incremented.

A nested if will determine the actions of whether the queue will be reset (if the index of the array is greater than zero) or the period of the timer will be changed and reset (if the index of the array is less than 6) or if the program will terminate and timers deleted (if the index of the array is equivalent to 6).

## SenderCallB Task:

This function is designed to be the call back function for the first timer, SenderTimer. It releases the semaphore when called so the Sender Task can be accessed.

## ReceiverCallB Task:

This function is designed to be the call back function for the second timer, ReceiverTimer. It releases the semaphore when called as long as the received messages are less than 500, if greater, it instead calls the init function mentioned above.

## Sender Tasks:

A buffer is declared as an array of characters and an integer to store to the current time.

A Queue, containing two slots of size equal to that of the array of characters, is created.

Inside an infinite, the task attempts to take the semaphore of the sender and if it succeeds, it sends the current time to the queue and waits for the semaphore to be released to repeat.

The counter for the sent messages is incremented if the task was successful and the counted for failed ones increments if not.

## Receiver Task:

Like the previous function a buffer is declared as an array of characters to receive the current time.

Inside an infinite while loop, the tasks attempts to take the semaphore of the receiver. If possible the Task receives the value inside the queue and increments the received counter.

### The following flowchart describes the system design fully.

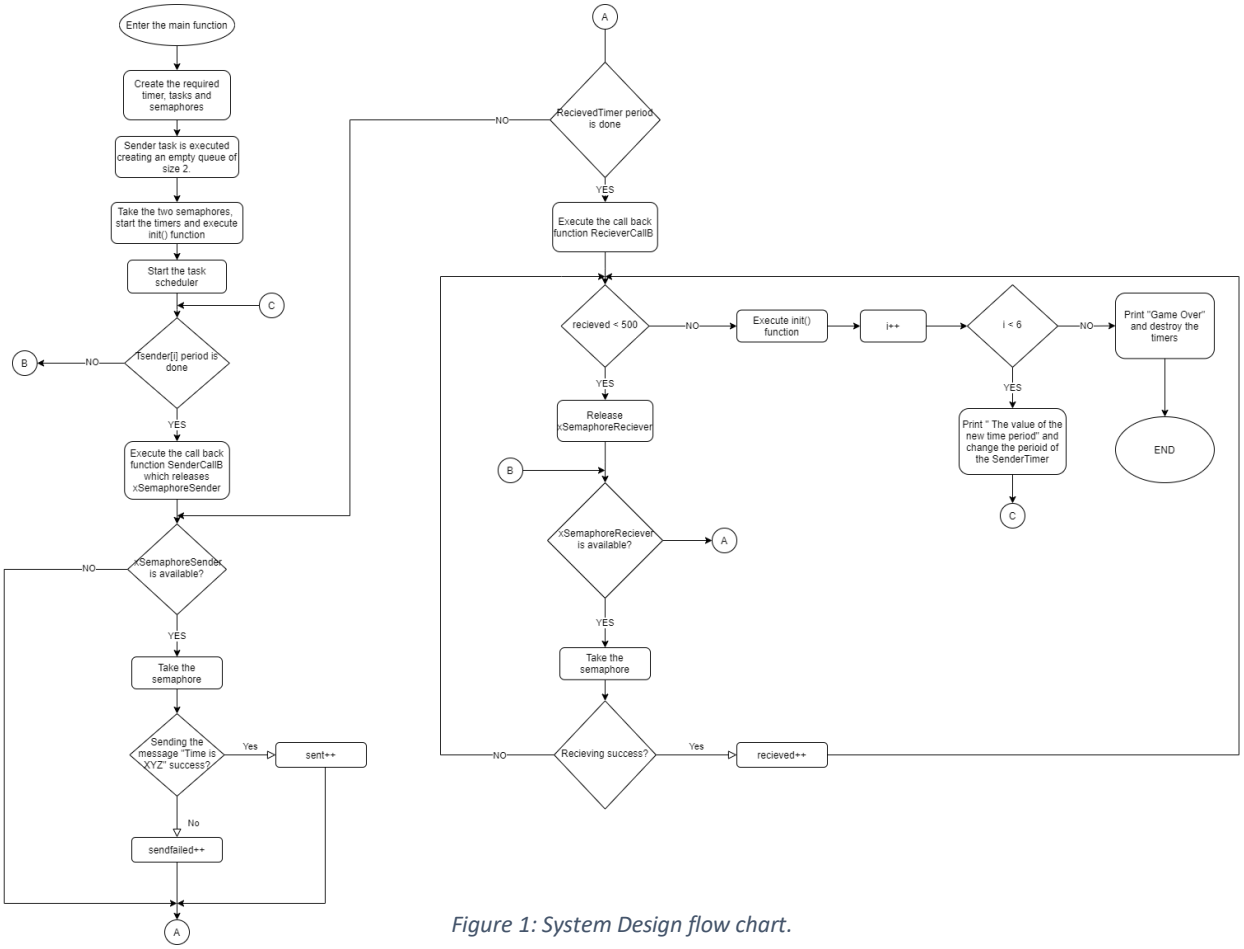


*Figure 1: System Design flow chart.*

# 2) Results

**The following Bar Chart shows the number of sent messages and the number of blocked messages as a function of Tsender Periods.**

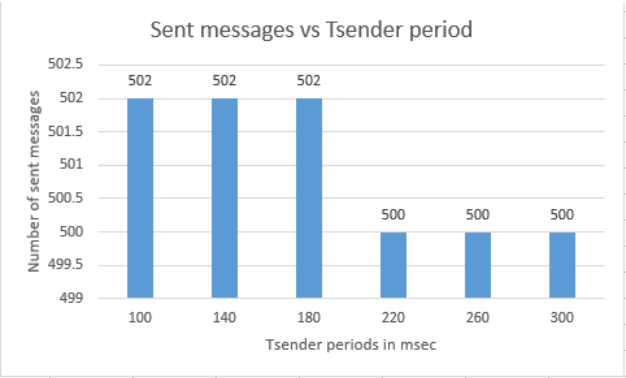


*Figure 2: Sent messages vs Tsender periods.*

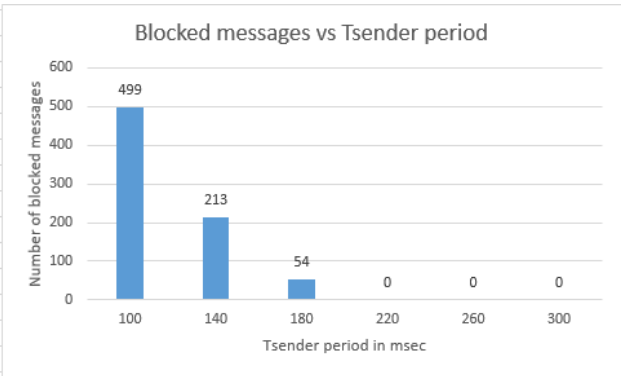


*Figure 3: Blocked messages vs Tsender periods.*

## Q1) The gap between the number of sent and received messages in the period.

The gap is present due to the low sender periods, less than 200 msec, having a higher frequency than that of the receiver causing messages to be blocked once the queue is full. So the number of sent messages will be equal to the maximum number of received messages (500) + the size of the queue.

At higher periods, more than 200ms, all sent messages are received so the queue is always empty; the number of sent messages will be equal to the maximum number of received messages (500) and the blocked messages = 0.

**Q2) Use a queue of size 2, then repeat for a queue of size 20. What happens when queue size increases?**

```
PRIGROUP unimplemented
The number of sent messages 0

The number of blocked messages 0

The new timer is : 100 msec

NVIC: Bad read offset 0xf34
NVIC: Bad write offset 0xf34
The number of sent messages 502

The number of blocked messages 499

The new timer is : 140 msec

The number of sent messages 502

The number of blocked messages 213

The new timer is : 180 msec

The number of sent messages 502

The number of blocked messages 54

The new timer is : 220 msec

The number of sent messages 500

The number of blocked messages 0

The new timer is : 260 msec

The number of sent messages 500

The number of blocked messages 0

The new timer is : 300 msec

The number of sent messages 500

The number of blocked messages 0

Game Over
```

*Figure 4: Output for queue of size 2.*

```
PRIGROUP unimplemented
The number of sent messages 0

The number of blocked messages 0

The new timer is : 100 msec

NVIC: Bad read offset 0xf34
NVIC: Bad write offset 0xf34
Other event 0x302
The number of sent messages 520

The number of blocked messages 481

The new timer is : 140 msec

The number of sent messages 520

The number of blocked messages 195

The new timer is : 180 msec

The number of sent messages 520

The number of blocked messages 36

The new timer is : 220 msec

The number of sent messages 500

The number of blocked messages 0

The new timer is : 260 msec

The number of sent messages 500

The number of blocked messages 0

The new timer is : 300 msec

The number of sent messages 500

The number of blocked messages 0

Game Over
```

*Figure 5: Output for queue of 20.*

The outputs for the different stages and queues are displayed above, one of size 2 and the other of 20. The number of sent messages increases and the number of blocked messages decreases as the queue size increases.

It's notable that when the size of queue increases, the number of blocked messages decreases since the queue has more space to receive the messages.

Still the sent messages @low periods = received_max + queue size

While sent messages @high periods = received_max.

This meets our conclusion from question 1.

# 3) References:


[1] The lectures' slides and recordings.

[2] FreeRTOS website