1 System Design

The given code is an example of a FreeRTOS program that demonstrates task communication and synchronization using queues, timers, and semaphores. Here is the overall flow of the program and how the tasks communicate and synchronize:

1. Include necessary header files and define constants and variables.
2. Implement the sender task function and receiver task function.
3. Implement the reset function where at its first call it only reset counters, clears the queue, and configures the values controlling the sender timer period.
4. Define 4 timer callback functions, one for each sender task and one for the receiver task.
5. In the main function:
    - Create a queue with a specified size that can be set from one of the constants that we defined at the start of the program.
    - Create 3 sender tasks (one for each sender) and the receiver task.
    - Create semaphores and timers for sender and receiver tasks (one for each).
    - Set the reset function to be called at the beginning and when the receiver task receives 1000 messages.
    - Start the FreeRTOS scheduler.
6. Implement FreeRTOS hooks and configurations.

Now let's analyze the program flow in more detail:

1. In the main function, after initializing variables and creating tasks, semaphores, and timers, the reset function is called to initialize the program state.
2. The 3 sender tasks are executed in parallel. Each task waits for its corresponding semaphore to be given, indicating it can proceed.
3. When a sender task gets the semaphore, it generates a random period of time based on the current iteration. It then changes the period of its corresponding timer and sends a message to the queue containing the current time.
4. The receiver task waits for the Receiver Semaphore to be given, indicating a message is available in the queue. It reads the message from the queue and increments the variable of the received Messages Count.
5. After a certain number of received messages (1000 in this case), the reset Function is called. It resets the counters, clears the queue, changes the periods of sender timers, and increments the iterations counter. If the iterations counter reaches 5, the program terminates(5 because the total number of random periods is 6 so by putting them in an array the last index is 5 while we start the iteration counter with a value of -1 to neglect the count of the reset function call inside the main so the iteration counter starts with 0 in the array & the reason that program terminates when the iteration counter reaches 5 not 6 is because in our code we increment the iteration counter variable after we check so if it is 5 it means it should begin next time with iteration variable of value 6 which is not available in the array so program terminates at iteration variable of value 5).
6. The timer callbacks for sender tasks and receiver task are executed when the corresponding timers expire. They give the corresponding semaphores to unblock the tasks or trigger the receiver task.
7. The program continues to run until it is terminated by reaching the end of the main function (should never do that) or by a termination condition specified in the code & before it ends it destroys all the timers.
8. a brief overview of the tasks and their priorities in the provided code:

Sender Task 1: This task has a priority of 2.
Sender Task 2: This task has a priority of 1.
Sender Task 3: This task has a priority of 1.
Receiver Task: This task has a priority of 3.

Where the higher the number is the higher the priority of the task.

In summary, the program consists of three sender tasks that generate messages at random intervals and a receiver task that receives messages from the queue. The tasks communicate and synchronize using semaphores and timers. The program collects statistics on sent, blocked, and received messages and resets them periodically.

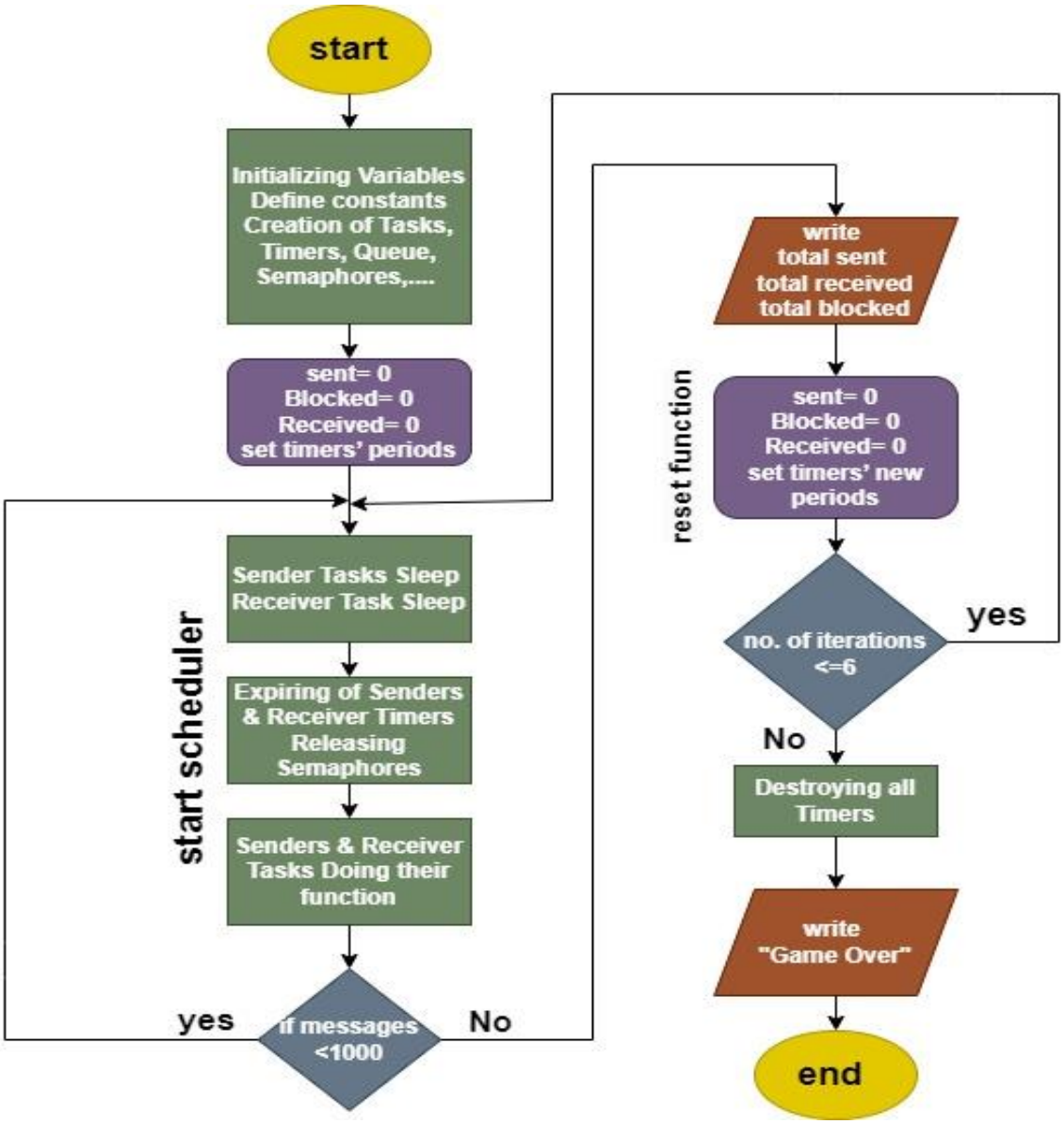
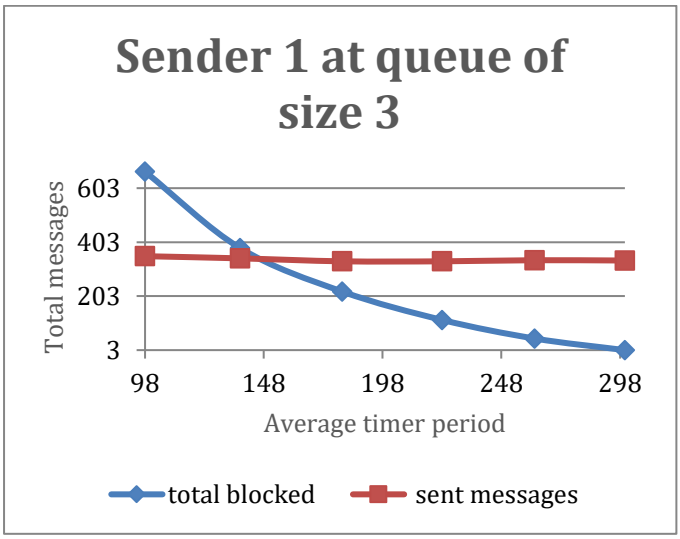
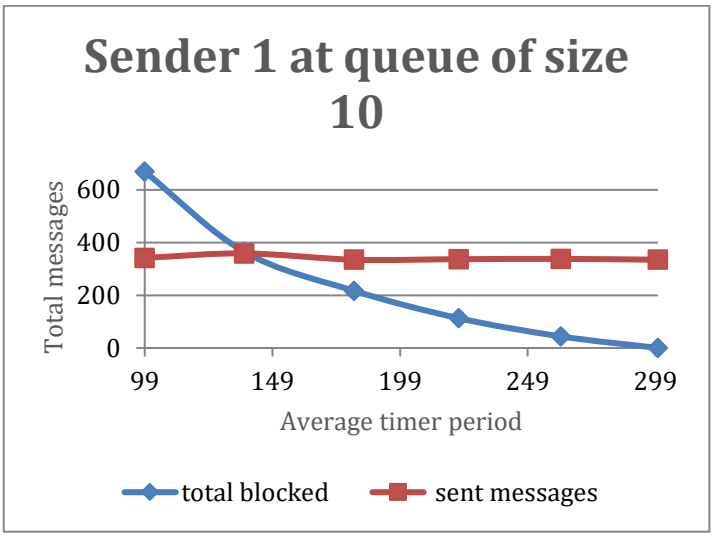


Figure 1: Flow chart

Please note that this representation is a simplified summary of the code's flow, and some details may be omitted. A comprehensive flowchart would provide a more detailed visual representation of the program's control flow, including conditionals, loops, and function calls.

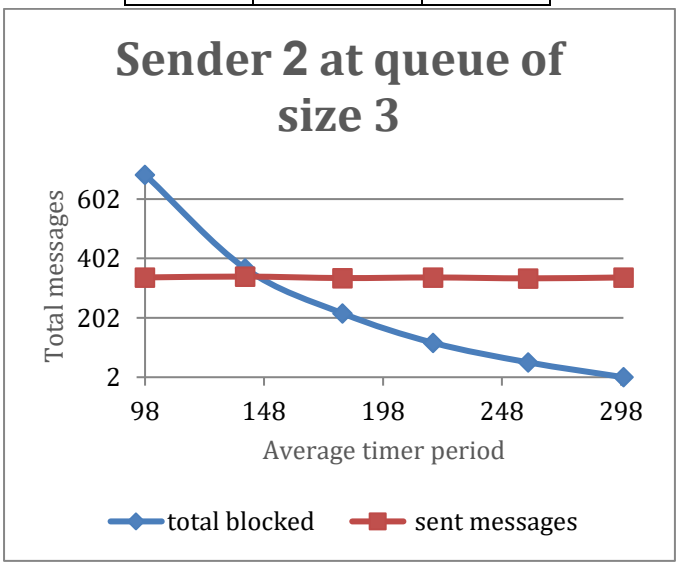
2 Results and Discussion

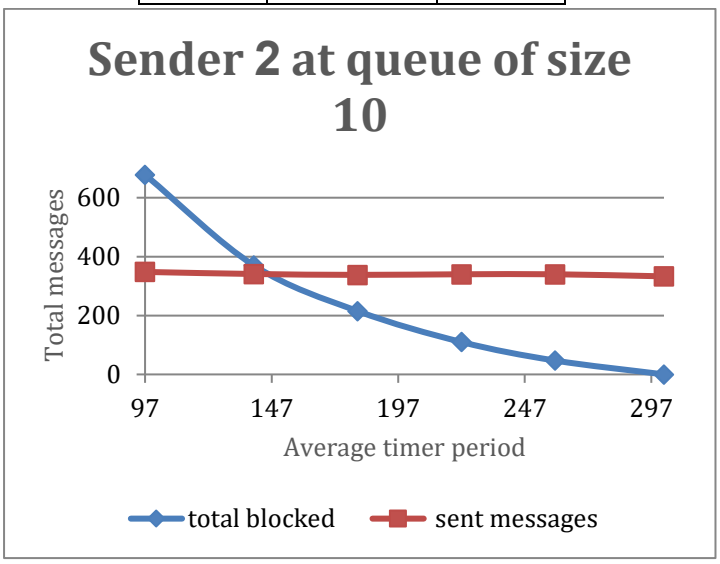




Average timer period 1	Sent successfully	Blocked
98	351	664
138	343	381
181	332	220
223	332	115
262	336	46
300	335	3

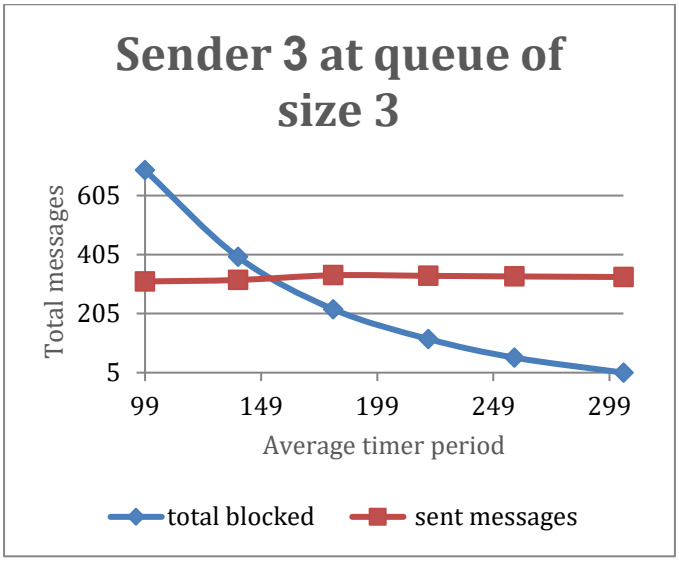
Average timer period 1	Sent successfully	Blocked
99	342	669
138	359	365
181	335	217
222	337	113
262	338	44
300	335	0

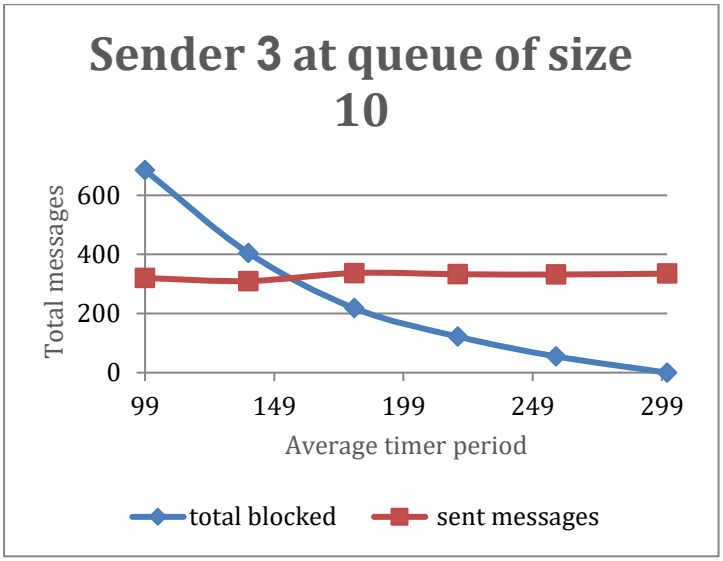




Average timer period 2	Sent successfully	Blocked
98	338	684
140	341	369
181	336	217
219	338	118
259	335	52
299	338	2

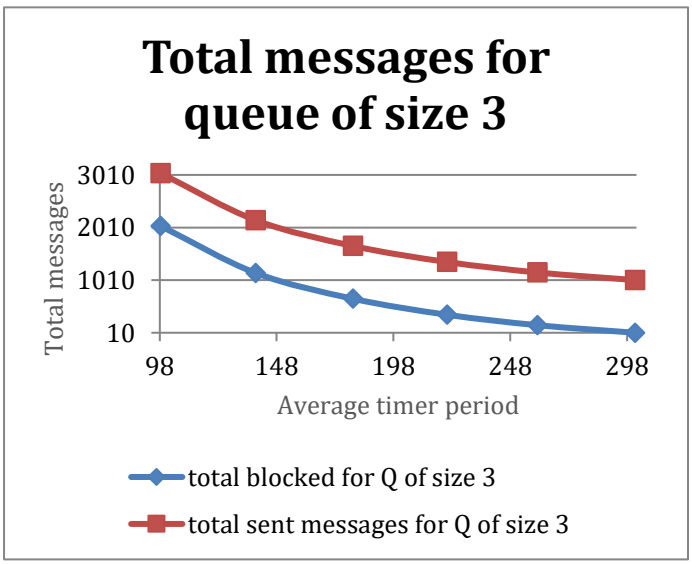
Average timer period 2	Sent successfully	Blocked
97	348	678
140	341	370
181	338	215
222	340	110
259	340	47
302	333	0

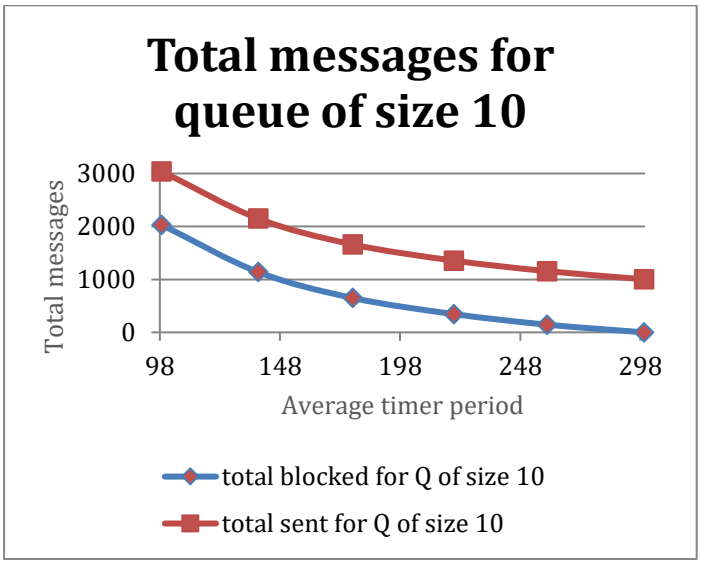




Average timer period 3	Sent successfully	Blocked
99	314	691
139	319	397
180	335	220
221	333	119
258	331	56
305	329	5

Average timer period 3	Sent successfully	Blocked
99	320	685
139	310	405
180	337	218
220	333	122
258	332	55
301	335	0





Average timer period	Total blocked	Total sent
98.333	2039	3042
139	1147	2150
180.667	657	1660
221	352	1355
259.667	154	1156
301.33	10	1012

Average timer period	Total blocked	Total sent
98.667	2032	3042
138.87	1140	2150
178.23	650	1660
220.37	345	1355
259.11	146	1156
299.45	0	1003

From the above results, we can see that as the average timer period of the task increases the number of blocked messages decreases and the successfully sent messages are nearly constant this makes sense as the time of sender increases this means that the rate that it sends with it is becoming slower while the receiver time is constant so this leads to less blocked messages as in this added time the receiver can receive some more messages so queue would have more empty space to receive from the senders.

We also see that when the queue is 10, the number of blocked messages becomes lower as the queue would have more space.

About the gap, we can find that the total sent messages successfully can be up to 1003 while the number of received messages is 1000 in case of a queue of size 3 and that number can become up to 1010 in case of a queue of size 10, this gap between the total successfully sent messages and the received messages is the number of messages that are still in the queue that were sent successfully to the queue but didn't become received by the receiver as the receiver called the reset once it received 1000 messages.

While the gap between the total sent messages (successfully sent messages +blocked messages) and the received messages is due to the blocked messages which are blocked because the queue is full and can't receive anymore at the time which sender sent its message.

3 References

[1] flow chart maker & online diagram software [Online] Available: https://app.diagrams.net/

[2] FreeRTOS, [Online] Available: https://www.freertos.org/a00106.html

[3] Chat gpt : https://chat.openai.com/