**A**

**1. What is stored within each message object passed into the queue for this program?**

Each message object stores a MessageID and a timestamp.

**2. Why can the two receive tasks read from their queues without using a mutex?**

Receive task are high priority so they will be take a message from the Queue as soon as it is ready to be send. However, the tasks can communicate without a mutex since the receive functions are blocked until a message in the queue is ready to be sent. As such when the Receiver function pre-empts the sender function it wont matter as the information it will be accessing is completely different. It will only access the shared information from the sender task after it has been fully added to the queue

* **Receiver 1**: The message is passed by value, meaning it is copied into the queue. Each task gets its own copy, so concurrent access does not cause any issues. Even if the original variable is modified elsewhere, the task reads the unchanged copy.
* **Receiver 2**: The message is passed by pointer, but only one task reads it, and as long as the data at the pointed-to memory address is not modified while being read, there's no need for a mutex. However, this approach can be risky if the data is changed elsewhere in the system.

**3. Where is the data stored in memory when we pass the message by value and where is it when we pass it by a pointer?**

* **By value**: The data is stored inside the queue. A complete copy of the message is made during the send operation.
* **By pointer**: Only the pointer is stored in the queue, while the actual data remains at its original memory address.

**4. Use uxQueueMessagesWaiting() inside one of the receive tasks and print its value. What numbers do you see while the program runs, and what do they mean?**

* With a delay before the receive task starts, the queue may show 4 messages waiting, indicating it has filled up before being read.
* Without the delay, the value is often 0, because the receive task removes messages immediately as they arrive, preventing queue buildup.
* Additionally, with the size of the queue the sender functions should always fine the queue empty.

The value returned by uxQueueMessagesWaiting() indicates the number of items currently stored in the queue, waiting to be received.

**5. Change the queue length from 4 to 1 in the example program. What changes in the behavior and why?**

* With a queue length of 1, only one message can be stored at a time. If the queue is full, sending tasks must wait until space is available.
* This introduces blocking behaviour in the sending tasks and may lead to a less predictable message order.
* With a length of 4, senders rarely block, and the output is more structured (e.g., 0,0,1,2,3...).
* With a length of 1, the program’s output can appear unpredictable, depending on task execution timing and scheduling.

**6. List two problems that can happen when we send a pointer instead of a copy, and one way to stop each problem.**

1. **Problem**: The original data is modified while still in the queue, causing race conditions or incorrect reads.
   * **Solution**: Use a mutex or semaphore to protect access until the receiver has finished reading.
2. **Problem**: The pointer becomes invalid or points to deallocated/moved memory, leading to undefined behaviour.
   * **Solution**: The receiver should copy the data into a local buffer immediately after receiving it, so the original memory can be safely reused or changed.

A

1. What is stored within each message object passed into the queue for this program?:

The MessageID and the time stamp

1. Why can the two receive tasks read from their queues without using a mutex? (0.25

Marks)

Receiver 1: Because it is being transferred by copy and is only read from the queue, therefore if it is interrupted it will simply continue reading. And even if the original value is changed, since it copied by value, it will just have old data instead of corrupted data.

Receiver 2: because it is being passed a pointer to the value, and unless the value in that point is changed being interrupted will not affect this.

1. Where is the data stored in memory when we pass the message by value and where

is it when we pass it by a pointer? (0.25 Marks)

Passed by memory the value is stored within the Queue, when it is passed via pointer it is stored in the original address.

1. Use uxQueueMessagesWaiting() inside one of the Receive tasks and print its value.

What numbers do you see while the program runs, and what do they mean? (0.25

Marks)

The number 4 in the queue repeatedly because we added a delay before allowing the queue to completely fill before reading. It is zero without the delay because it reads nothing in the queue at the start, then the task waits until an item is in the queue then takes and reads it, meaning by the time it gets back to the check the queue is still empty

1. Change the queue length from 4 to 1 in the example program. Write down what you

observe change in the behaviour of the program and explain why. (0.25 Marks)

The items in queue is only ever one because that is the maximum number of items that can be stored. Additionally, the order in which the send tasks are send is different as with a queue length of 4 and the receives tasks removing them once they are read, the send tasks should always find the queue empty. With a queue length of 1 certain send tasks will be blocked which distrupts the process

With a queue length of 4 it goes, (0,0,1,2,3,0,0,1,2,3,ect) with a queue length of 1 it become unpredictable

1. List two problems that can happen when we send a pointer instead of a copy, and give

one way to stop each problem. (0.25 Marks)

If the data in the original place is change

**If the data is read at the same time something is written to, or is changed there could be a race condition which returns garbage:**

Have a mutex or semaphore signal or stop data from read or changed before the second queue has finished being read.

**If the data in the original memory address is modified or moved, it will interfere with the access to the data for example if the data is dereferenced, will referencing a null pointer which will result in undefined behaviour:**

Once it has been received an assuming that the previous problem has prevented the original data being changed while in the Queue, the receiving task can read the data in the reference and copy it into a separate location in memory specifically for the receive task to use. So even if the sender task changes or dereference the original point the data will still be safe