

Queues

INTRODUCTION

A message queue is a list of data items that communicates information between two systems or processes, or between two tasks or threads in the same program. This application note is concerned with the basics of how queues are used to communicate data between tasks in a Real Time Operating System (RTOS).

QUEUES

A queue is a list of data items. Conceptually it could grow to any number of items, but in practice it is normally a fixed maximum length. The storage size of the data item is fixed, but the queue itself is not concerned with the nature of the data that is placed in it. The message could be a simple data variable or more structured.

The message queue enables processing of data to be asynchronous; the process that creates the data does not have to wait for the receiving process to be ready to receive the data. The receiving process can take the data out of the queue when it comes to the appropriate point in its execution.

Common Queue implementations include:

- First In First Out (FIFO) buffers. Data is written to the back of the queue and taken from the front of the queue. However, it is also possible to add data to the front of the queue or overwrite the data stored in the head or tail entry.
- Circular Buffers. The queue behaves as if the two ends were connected.
- A Linked List. This is a variant of a queue, where the list can be any length and data items can be inserted or removed at any point.

The data stored in the queue can be either copies of the data to be transmitted between tasks, or pointers to the data. Queuing by copy is copying the actual data of the message to the queue. Queuing by reference is to queue a pointer to the data. Queuing by reference can enable messages consisting of very large amounts of data to be passed. It is a more efficient use of RAM and processing time than Queuing by copy, but needs more careful programming to ensure that the referenced data exists so that the pointer remains valid until it has been communicated.

USING A QUEUE

Any task or Interrupt Service Routine (ISR) that knows of the existence of a queue can write to it or read from it. This means that multiple tasks can use it.

A task that is waiting to write to a queue can optionally be placed in the blocked state for a designated length of time. When space to write to is available in the queue, the task will become ready. If space does not become available, the task will become ready when the block time expires.

Similarly, a task that is waiting to read from a queue can be blocked for a specified length of time. The task will become ready when data appears in the queue, or when the block time expires, whichever is the earlier.

When multiple tasks are waiting on a queue, the highest priority is first to be unblocked, or, if the tasks are of equal priority, the task that has been waiting longest is the first to be unblocked.

It is common in microprocessor applications for one task to receive data items from multiple sources, and these items may be processed differently depending on where they came from. A single queue can be used to communicate structures that include both the data and information about the source of the data, so that the data can be processed in the appropriate way. It is also possible for a single task to be programmed to take data from multiple queues. However, a task can block only on one queue.

EXAMPLE OF A QUEUE

1. The queue is created empty. In this example, the queue can hold six items. It is used for passing messages from Task 1 to Task 2.



2. Task 1 writes the data of the message. The value written is now the only item in the queue, and is therefore the value at the back of the queue *and* the value at the front of the queue.



3. Task 1 writes a second message to the next position in the queue. Message A is still at the front of the queue, but Message B is now at the back. There are four empty spaces remaining in the queue.



4. Task 2 is now ready to read (receive) the first message. This is taken off the front of the queue.



5. The next time Task 2 reads from the queue it will receive message B as this is now at the front of the queue.



QUEUE API

An operating system API will include functions for creating:

- Queues;
- Tasks to send data items to the front or the back of a queue;
- Tasks to read an item from the front of the queue;
- Tasks to read an item from the front of the queue and remove it;
- And for checking how many items are waiting in the queue.

CONSIDERATIONS WITH QUEUES

Use of message queues requires a number of implementation decisions on the part of the engineer. There is a trade-off between memory and performance; setting the queue length at a higher value increases throughput at the expense of greater memory requirement.

Similarly, the maximum size of the data to be transferred determines the fixed size of the queue item and therefore the memory required.

SAFERTOS® AND QUEUES

SAFERTOS® provides the capability to send data to either the front or the back of a queue, enhancing its flexibility. By allowing data to be sent to the front, high-priority items can be prioritized accordingly. Additionally, if items are consistently sent to the front of the queue, it will function as a stack, offering versatile data management options.

WITTENSTEIN high **integrity** systems

Worldwide Sales and Support

Americas: +1 408 625 4712

ROTW: +44 1275 395 600

Email: sales@highintegritysystems.com

Web: www.highintegritysystems.com



WITTENSTEIN