Mailbox Introduction

Queues are the linear data structure used for inter-process data communication such as sharing data between tasks or interrupt services routines. Sender task writes data to the queue and the receiver task reads data from the queue and after read operation data item removes from the queue.

Unlike traditional queues, a mailbox holds the data that can be read by any task, and data remains in the mailbox until overwrites by another task.

In other words, In the case of a mailbox, a writes task sends data to the queue and a receiver task reads data from the queue but does not remove it until it is overwritten by a sender task.

How to use FreeRTOS queue API to Create Mailbox

FreeRTOS queue management API also provides function which can be used to create a mailbox. These are the two functions:

FreeRTOS xQueueOverwrite() API

Like xQueueWrite() API function, xQueueOverwrite() also used to write data to a queue. But if queue is full xQueueWrite() enters the blocking state. On the contrary, xQueueOverwrite() overwrites the data without going to blocking state.

Note: xQueueOverwrite() must used with a queue of length one only.

Furthermore, the input arguments to this function is same as xQueueWrite() function. More information about this link is available in this link:

Note: Never call xQueueOverwrite() from an interrupt service routine. The interrupt-safe version xQueueOverwriteFromISR() should be used in its place.

FreeRTOS xQueuePeek() API

Like xQueueRead() API function, xQueuePeek() is also used to read data from a queue. But unlike xQueueRead(), xQueuePeek() function does not delete data from the queue after reading it. This function does not remove the item from the queue or not modify it.

Note: Never call xQueuePeek() from an interrupt service routine. The interrupt-safe version xQueuePeekFromISR() should be used in its place.

FreeRTOS Mailbox Example using Arduino

Now let's see an example of a mailbox with Arduino. In this example, we create two tasks, one task writes integer value to the mailbox after every delay of vTaskDelay(500) and vReadMailbox() task reads that integer value from the mailbox after every delay of vTaskDelay(100). This means vUpdateMailbox() overwrite mailbox data one time as compare to vReadMailbox() that will receive it five times. Because, task delay time of vUpdateMailbox() is five times of vReadMailbox() task.

Arduino Example Code

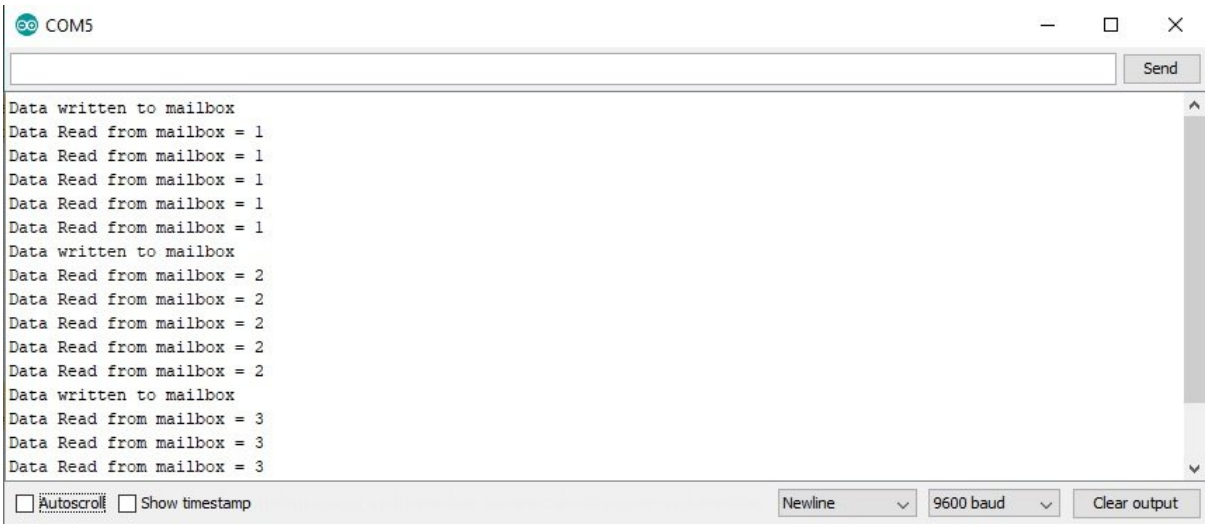
```
#include <Arduino_FreeRTOS.h>

#include <queue.h>



QueueHandle_t    xMailbox;

TaskHandle_t      TaskHandle_1; // handler for Task1

TaskHandle_t      TaskHandle_2; // handler for Task2
```

```
void setup() {
 // put your setup code here, to run once:
 Serial.begin(9600);
 xMailbox = xQueueCreate(1, sizeof( int32_t));
// Serial.begin(9600); // Enable serial communication library.
 xTaskCreate(vUpdateMailbox, "Sender", 100, NULL, 1, &TaskHandle_1);
 xTaskCreate(vReadMailbox, "Receiver", 100, NULL, 1, &TaskHandle_2);
}
void loop()
{
 // put your main code here, to run repeatedly:
}
void vUpdateMailbox(void )
{
int32_t ulNewValue = 1;
while(1)
{
xQueueOverwrite( xMailbox, &ulNewValue);
Serial.println("Data written to mailbox");
ulNewValue++;
vTaskDelay(500);
}
}

BaseType_t vReadMailbox(void )
{
```

```
int32_t value_received;

while(1)

{

xQueuePeek( xMailbox, &value_received, portMAX_DELAY );

Serial.print("Data Read from mailbox = ");

Serial.println(value_received);

vTaskDelay(100);

}

}
```

Program Output

As you can see from the output of the serial monitor, the reader task keeps reading the same data from the mailbox until the data update task does not overwrite the mailbox value. It also shows that unlike the FreeRTOS queue, data from the mailbox does not remove on a read operation until it is overwritten by a sender task.



Interrupt Management Introduction

While using RTOS, it is very critical to handle interrupt service routines. Because the misuse of interrupts can lead to time constraint issues such as other periodic tasks failing to meet their deadlines.

Note: Interrupts have higher priorities than other Tasks. Therefore, it Interrupts should not wait for a mutex, semaphore, and other resources and should be executed as soon as it occurs. Otherwise, it may cause issues. Defer processing of interrupts through other tasks is a possible countermeasure to minimize the processing time of ISR as soon as possible.

" In short, the code and execution time of interrupt service routine should be as small as possible."

Name of the interrupts	Interrupt Number	ISR Location
Einta	2	Ox5600
Eintb	5	0x5700
Eintc	8	0x5800
eintd	1	0x5900

RAM --- data

ROM -- code

EEPROM -- Flash Memory

Stack Memory --- ISR

Cache Memory

TASK (Read a data from the queue) → ISR(put a data in the queue)

Using Interrupt to Read and Write to Queues

FreeRTOS interrupt provides an interrupt safe version of queue API to read and write data from queues using ISR. These are the two API functions:

- xQueueSendToBackFromISR() : It is used to write data to the queue from an interrupt service routine. It works similar to xQueueSendToBack() API with the same function of input argument and a return value. For information about this API, check this link.
- xQueueReceiveFromISR(): It is used to read data from the queue from an interrupt service routine. It works similar to xQueueReceive() API with the same function of input argument and a return value.

FreeRTOS Interrupt Management Example with Arduino

In this example, we will create a task to print a string on an Arduino Serial monitor. First, it will read a string from a string type queue.

An interrupt service routine writes the string to the queue and the print function reads value from that queue and prints it on the serial monitor of Arduino.

To generate interrupt, we will use a timer1 of Arduino. Timer1 of Arduino will generate an interrupt on every overflow. That means, the interrupt service routine will execute on overflow of timer1.

Inside the ISR, we define 5 strings and depending on the string number passed to the xQueueSendToBackFromISR() function , this API writes corresponding string to the queue string and string print task display message on Arduino serial monitor accordingly.

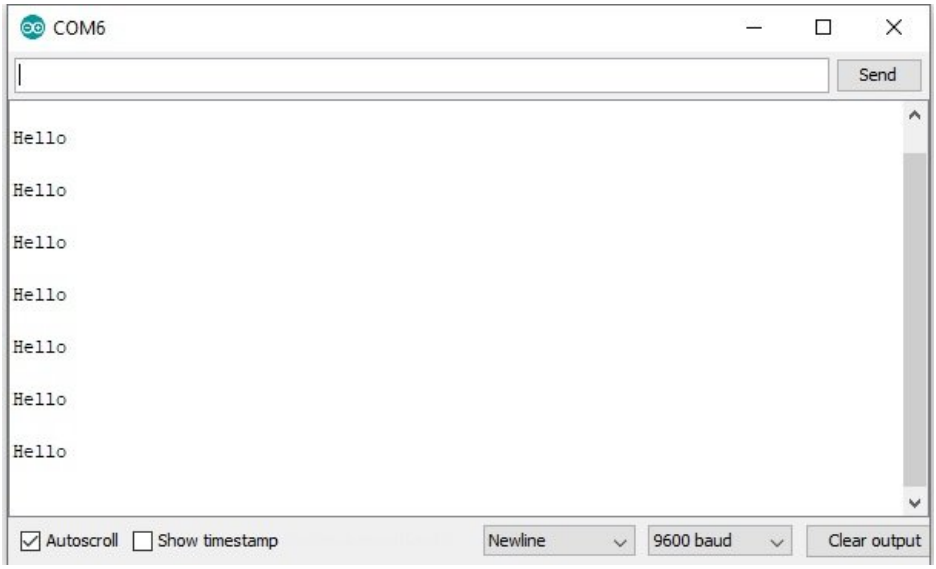
Arduino Code

Copy this example code and upload it your Arduino board. First lets check the output of this code and after that we will explain the working of code.

```
#include <Arduino_FreeRTOS.h>

#include "queue.h"



QueueHandle_t      xStringQueue;

int timer1_counter ;

void setup()

{

   Serial.begin(9600);

 xTaskCreate(vStringPrinter," String Printer", 100, NULL, 1,NULL);

 xStringQueue  = xQueueCreate(5,sizeof(char *));

 InterruptInit();

}

void vStringPrinter(void *pvParameters)

{

 char *pcString;

 while(1)

 {
```

```
  xQueueReceive(xStringQueue,&pcString, portMAX_DELAY);
  Serial.println(pcString);
 }
}
ISR(TIMER1_OVF_vect){
 TCNT1 = timer1_counter;
 uint32_t receivedNumber;
 static const char *pcStrings[]=
 {
  "Hello\r\n",
  "Hi\r\n",
  "I\r\n",
  "am\r\n",
  "here\r\n",
 };
 xQueueSendToBackFromISR(xStringQueue,&pcStrings[0],pdFALSE);
 }
void loop(){}


void InterruptInit()    // critical
{
 noInterrupts();           // disable the interrupts
 TCCR1A =0;        // initializing the interrupt and enable the interrupt
 TCCR1B =0;
 timer1_counter = 34286;
 TCNT1 = timer1_counter;
 TCCR1B |=(1<<CS12);
 TIMSK1 |= (1 << TOIE1);
```

```
interrupts();             // enable the interrupts
}
```



Example:

```
#include <Arduino_FreeRTOS.h>
#include "queue.h"

QueueHandle_t xStringQueue;
int timer1_counter ;
void setup()
{
Serial.begin(9600);
  xTaskCreate(vStringPrinter," String Printer", 100, NULL, 1,NULL);
  xStringQueue  = xQueueCreate(5,sizeof(char *));
  InterruptInit();
}

void vStringPrinter(void *pvParameters)
{
  char *pcString;
  while(1)
  {
   //  Serial.println("Inside the Task1");
    xQueueReceive(xStringQueue,&pcString,portMAX_DELAY);
    Serial.println(pcString);
  }
}
```

```
ISR(TIMER1_OVF_vect)
{
  TCNT1 = timer1_counter;
  uint32_t receivedNumber;
  static const char *pcStrings[]=
  {
    "WELCOME\r\n",

  };
  xQueueSendToBackFromISR(xStringQueue,&pcStrings[0],pdFALSE);
  }

void loop()
{

}

void InterruptInit()
{
  noInterrupts();
  TCCR1A =0;
  TCCR1B =0;
  timer1_counter = 34286;
  TCNT1 = timer1_counter;
  TCCR1B |=(1<<CS12);
 TIMSK1 |= (1 << TOIE1);
  interrupts();
}
```