

QUEUES NEDİR RTOS' DA HANGİ AMAÇLARLA KULLANILIR?

Temelde kuyuklar sınırlı sayıda sabit boyutlu veri ögesi tutabilen bir veri yapısıdır.



RTOS' da ise görevler arasında veri ve bilgi paylaşmasını ve görevler arasındaki etkileşimi sağlamak için kullanılır. Kuyuklar, görevler arasında veri transferi yapmak için kullanılan ara yüzlerdir.

Bir RTOS kuyruğu, belirli bir tür veriyi depolamak için tasarlanmış bir veri yapısıdır. Görevler, kuyruğa veri yazabilir ve diğer görevler tarafından okunabilir. Bu şekilde, farklı görevler arasında veri paylaşımı yapılabilir. Kuyuklar, bir görevin diğer görevlere veri aktarmasını ve diğer görevlerin veriyi okumasını sağlar.

RTOS kuyukları, veri transferinin hızlı ve güvenli bir şekilde yapılmasını sağlar. Ayrıca, kuyuklar sayesinde görevler arasında etkileşimin düzenlenmesi ve görevlerin veri beklemede kalmasının önlenmesi mümkündür. Bu sayede, RTOS, görevler arasında veri transferi yapmak için kullanılan bir ara yüz sağlar ve görevler arasındaki etkileşimi kolaylaştırır.

FreeRTOS' DA KUYRUK OLUŞTURMA VE BU KUYRUKTAN VERİ GÖNDERME VE VERİ ALMA UYGULAMASI

Bu projede 2 adet görev olacaktır.

Bu görevler: **SenderTask** ve **ReceiverTask**

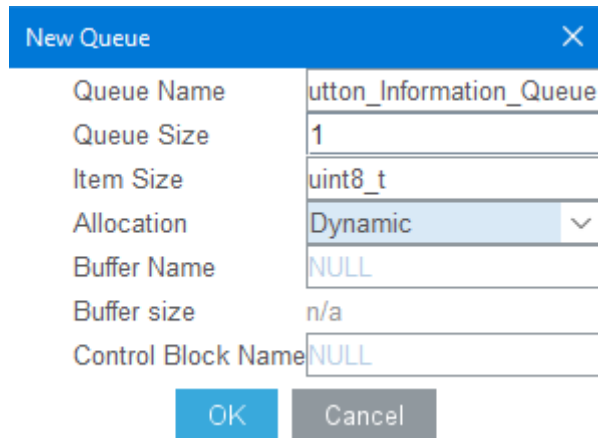
SenderTask: Bu görevde Butonun durumu kuyruğa yazılacak.

ReceiverTask: Bu görevde Kuyrukta olan mesaja göre led yanıp sönecek.

Edit Task	
Task Name	SenderTask
Priority	osPriorityNormal
Stack Size (Words)	128
Entry Function	SenderTaskFunc
Code Generation Option	Default
Parameter	NULL
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Edit Task	
Task Name	ReceiverTask
Priority	osPriorityNormal
Stack Size (Words)	128
Entry Function	ReceiverTaskFunc
Code Generation Option	Default
Parameter	NULL
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Ardından **Button_Information_Queue** adında kuyruk yapısını oluşturuyoruz.



Ardından CTRL + S tuşlarına basıp otomatik olarak kodların oluşturulmasını bekliyoruz.

Kuyruk oluşturma ile ilgili kodlar aşağıdaki gibi otomatik oluşturulduğundan bizim işlem yapmamıza gerek kalmıyor.

```
/* Create the queue(s) */
/* definition and creation of Button_Information_Queue */

osMessageQDef(Button_Information_Queue, 1, uint8_t);
//osMessageQDef() bu fonksiyon sayesinde Kuyruk yapısı için hafızadan yer ayrıldı.
//3 adet parametre alır bunlar.
//1) kuyruk adı.
//2) Bir kuyrukta bulunabilecek max mesaj sayısı.
//3) kuyruktaki elemanların veri tipi

Button_Information_QueueHandle = osMessageCreate(osMessageQ(Button_Information_Queue), NULL);
//osMessageCreate(): bu fonksiyon sayesinde kuyruk kullanıma hazır hale geldi
//2 adet parametre alır
//1) Bu parametre bir makrodur ve verilen parametrenin referansını döner (&Button_Information_Queue)
//2) Bu parametre osThreadId (TaskHandle_t) türünden değişkenleri alır
```

SenderTaskFunc fonksiyonumuzu aşağıdaki gibi dolduruyoruz:

```
/* USER CODE END Header_SenderTaskFunc */
void SenderTaskFunc(void const * argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        if(!HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13))
        {
            osMessagePut(Button_Information_QueueHandle, 1, osWaitForever);
            while(!HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13));
        }
        else
        {
            osMessagePut(Button_Information_QueueHandle, 0, osWaitForever);
            while(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13));
        }

        osDelay(1);
    }
    /* USER CODE END 5 */
}
```

ReceiverTaskFunc fonksiyonumuzu aşağıdaki gibi dolduruyoruz:

```
/* USER CODE END Header_ReceiverTaskFunc */
void ReceiverTaskFunc(void const * argument)
{
    /* USER CODE BEGIN ReceiverTaskFunc */
    /* Infinite loop */
    for(;;)
    {
        osEvent Event;
        //osMessageAvailableSpace fonksiyonu kuyrukta ne kadar yer kaldığını döndürür.
        if(osMessageAvailableSpace(Button_Information_QueueHandle) == 0)
        { //Kuyrukta yer kalmadıysa mesaj gelmiştir demektir.
            Event = osMessageGet(Button_Information_QueueHandle, osWaitForever);
            if(Event.status == osEventMessage) // kuyruğa mesaj geldiyse aşağıya in
            {
                if(Event.value.v == Button_Pressed) // buton basıldı
                {
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
                }
                else // buton serbest bırakıldı
                {
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
                }
            }
        }
        osDelay(1);
    }
    /* USER CODE END ReceiverTaskFunc */
}
```

Butona basıldığında SenderTask kuyruğa 1 değerini yazacak ReceiverTask ise kuyruğa yazılan 1 değerini okuyup ledi yakıcak

Buton serbest bırakıldığında SenderTask kuyruğa 0 değerini yazacak ReceiverTask ise kuyruğa yazılan 0 değerini okuyup ledi söndürecek.

Projenin videosunu github’ da bulabilirsiniz.

Not:

Bu projede Queue’ yi tanımlamak için ayrıca xCreateQueue() API’ sinide kullanabiliriz.

```
/* Create the queue(s) */
/* definition and creation of Button_Information_Queue */

/*
osMessageQDef(Button_Information_Queue, 1, uint8_t);
Button_Information_QueueHandle = osMessageCreate(osMessageQ(Button_Information_Queue), NULL);
*/

/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
Button_Information_QueueHandle = xQueueCreate(1, sizeof(uint8_t));
//xQueueCreate() 2 adet parametre alır
// 1) Kuyruğun uzunluğu
// 2) Kuyruktaki elemanların boyutu
/* USER CODE END RTOS_QUEUES */
```

